An investigation of multigrid algorithms for a higher order Discontinuous Galerkin RANS solver

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines

Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von
Marcel Wallraff
geboren am 17.11.1983
in Köln.

Eingereicht am : 16.01.2015
Disputation am : 15.07.2015

1. Referent: Prof. Dr. Hermann G. Matthies
2. Referent: Prof. Francesco Bassi

2015

Abstract

(Published in english)

An investigation of multigrid algorithms for a higher order Discontinuous Galerkin RANS solver.

This thesis deals with the development of robust, efficient and scalable solver algorithms for a higher-order Discontinuous Galerkin discretization of steady-state turbulent flow problems in aerodynamic applications. The focus is laid on nonlinear and linear p- and h-multigrid methods in various combinations. A framework is developed which allows to compute reliable numerical results for flow problems in two and three dimensions faster than a state of the art single level solver.

Discontinuous Galerkin Method, Multigrid Method, Reynolds-Averaged Navier-Stokes Equations

Zusammenfassung

Eine Untersuchung von Mehrgitterverfahren im Rahmen eines diskontinuierlichen Galerkin Lösers für die Reynolds-gemittelten Navier-Stokes-Gleichungen.

In dieser Arbeit werden robuste, effiziente und skalierbare Algorithmen entwickelt zur Lösung von diskontinuierlichen Galerkin-Diskretisierungen höherer Ordnung von stationären turbulenten Strömungsproblemen in der aerodynamischen Anwendung. Das Hauptaugenmerk liegt auf der Entwicklung und Kombination von nichtlinearen und linearen p- und h-Mehrgitterverfahren. Die entwickelte Löserstruktur erlaubt es, verläßliche numerische Ergebnisse für Strömungsprobleme in zwei und drei Raumdimensionen mit Mehrgitter schneller als ohne Mehrgitter zu berechnen.

Publications

Some of the text fragments and figures presented in this thesis are already published before hand in:

[76] M. Wallraff, T. Leicht, and M. Lange-Hegermann. Numerical flux functions for Reynolds-averaged Navier-Stokes and $k\omega$ turbulence model computations with a line-preconditioned p-multigrid discontinuous Galerkin solver. *International Journal for Numerical Methods in Fluids*, 71(8):1055–1072, 2013

[75] M. Wallraff and T. Leicht. Higher order multigrid algorithms for a discontinuous Galerkin RANS solver. *52nd AIAA Aerospace Sciences Meeting, AIAA Paper 2014-0936*, 2014.

[74] M. Wallraff and T. Leicht. 3D application of higher order multigrid algorithms for a RANS-$k\omega$ DG-solver. In R. Abgrall, H. Beaugendre, P. M. Congedo, C. Dobrzynski, V. Perrier, and M. Ricchiuto, editors, *High Order Nonlinear Numerical Schemes for Evolutionary PDEs - Proceedings of the European Workshop HONOM 2013 , 18-22nd March, Bordeaux*, volume 99 of *Lecture Notes in Computational Science and Engineering*, pages 77–88. Springer, 2014.

[73] M. Wallraff, R. Hartmann, and T. Leicht. Multigrid solver algorithms for DG methods and applications to aerodynamic flows. In N. Kroll, C. Hirsch, F. Bassi, C. Johnston, and K. Hillewaert, editors, *IDIHOM - Industrialisation of High-Order Methods - A Top Down Approach*, volume 128 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 153–178. Springer, 2015.

Acknowledgment

This work has benefited from the support of many people I would like to thank in the following.

First, I would like to thank Prof. Hermann G. Matthies for supervising my thesis and his constructive comments. Equally, I'm grateful to Prof. Francesco Bassi for being the co-examiner of my thesis.

Furthermore, I express thanks to the department head of $C^2A^2S^2E$, Prof. Norbert Kroll, who offered me the opportunity to carry out my Ph.D work in the DLR Institute of Aerodynamics and Flow Technology. This work profited from many discussions with him and his very helpful comments and suggestions.

I am most grateful to Tobias Leicht and Dr. Ralf Hartmann and I would like to thank them for their professional competence, a lot of discussions, comments and suggestions and the proof-reading of this thesis.

Special thanks goes to Stefan Schoenawa and Christian Lenz for the proof-reading of this thesis and many helpful discussions. I am also thankful to my colleagues at $C^2A^2S^2E$ for the nice and relaxed atmosphere at work, especially to Niklas and Thomas.

My family and friends also deserve special thanks for their unconditional support and the necessary counter balance to my work.

Finally, I would like to acknowledge the financial support from the German "Bundesministerium für Bildung und Forschung (BMBF)" through the "Verbundprojekte 2010-2013 Mathematik für Innovationen in Industrie und Dienstleistungen", in particular the project "Diskontinuierliche Galerkin-Verfahren für den robusten aerodynamischen Entwurf auf effizienten Rechnerarchitekturen in der Luftfahrt (DGHPOPT)".

Contents

1 Introduction **1**

1.1 Literature review and contribution to the field 3

1.2 Objective 6

1.3 Thesis outline 6

2 Governing equations in CFD **8**

2.1 The compressible Navier-Stokes equations 8

2.2 Flux formulation of the steady-state Navier-Stokes equations 10

2.3 Reynolds-averaged Navier-Stokes equations 14

2.3.1 The steady-state RANS-$k\omega$ equations 15

2.3.2 The steady-state RANS-SA equations 19

3 Discontinuous Galerkin discretization **22**

3.1 Triangulation 22

3.2 Numerical flux function for convective terms 27

3.2.1 Roe flux 27

3.2.2 Eigenvalue decomposition of the flux Jacobian 28

3.3 Boundary conditions 30

3.4 Basis functions 32

3.5 Mesh related issues 33

4 Solver methods **36**

4.1 Nonlinear multigrid methods 39

4.2 Nonlinear smoothers / single level solver 40

4.2.1 Backward–Euler 41

4.2.2 GMRes method 43

4.3 Linear multigrid algorithm . . . 44
4.4 Linear smoothers / preconditioner . . . 45
4.5 Startup strategy . . . 47
4.6 Transfer operators . . . 49
4.7 Line creation . . . 51
4.8 Mesh agglomeration . . . 55
4.9 Solver overview . . . 58

5 Computational test cases **67**
5.1 Test case introduction and framework verification . . . 67
5.1.1 L1T2 high-lift three element airfoil . . . 68
5.1.2 MDA 30P30N three element airfoil . . . 74
5.1.3 VFE-2 delta-wing . . . 78
5.1.4 NASA Trap Wing . . . 82
5.2 Algorithmic investigations . . . 86
5.2.1 General settings and information . . . 88
5.2.2 Investigation of startup strategies . . . 92
5.2.3 Trade-off between nonlinear and linear iterations . . . 101
5.2.4 Coarse level linearization . . . 105
5.2.5 Linear multigrid algorithm as a preconditioner . . . 107
5.2.6 Linear multigrid algorithm as a solver . . . 116
5.2.7 Investigation of the nonlinear multigrid algorithm . . . 119
5.2.8 Investigation of different solver choices . . . 125
5.2.9 h- and p-independency . . . 131
5.3 Framework benchmark . . . 135

6 Conclusions **140**

A Convective flux Jacobian **143**

B TauBench **145**

Bibliography **147**

List of Notations

General notations

$\mathbf{u}$	vector in $\mathbb{R}^n$
$\mathbf{A}$	matrix in $\mathbb{R}^{n\times m}$
u_i	i-th entry of the vector $\mathbf{u}$
$(\mathbf{A})_{ij} = a_{ij}$	entry in the i-the row and the j-the column of the matrix $\mathbf{A}$
∇	vector of partial differential operators for each space dimension $(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_d})^\top$
u_∞, $\mathbf{u}_\infty$	far field values of u and $\mathbf{u}$ at Γ_{in} and Γ_{out}
u_Γ, $\mathbf{u}_\Gamma$	values of u and $\mathbf{u}$ at the wall Γ_{wall}
$\mathbf{u}^+$	value of $\mathbf{u}$ on a face inside an element
$\mathbf{u}^-$	value of $\mathbf{u}$ on a face outside an element
$\mathbf{I}$	identity matrix

Mathematical operators

$\mathbf{u}\cdot\mathbf{v}$	$\sum_{i=1}^{n} u_i v_i$	scalar product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$
$\mathbf{A} : \mathbf{B}$	$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij} b_{ij}$	double scalar product (Frobenius inner product) of two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times m}$
$\mathbf{u}\otimes\mathbf{v} =: \mathbf{A}$	$u_i v_j =: a_{ij}$	dyadic product of two vectors in $\mathbb{R}^n$
$\langle f, g\rangle_{L^2}$	$\int_\Omega f g \,\mathrm{d}V$	L^2 scalar product
δ_{ij}	$(\mathbf{I})_{ij}$	Kronecker delta
$D^\alpha u$	$\frac{\partial^{\lvert\alpha\rvert} u}{\partial x_1^{\alpha 1}\ldots\partial x_n^{\alpha n}}$	α-th weak derivative of u
$\{\!\{\mathbf{v}\}\!\}$	$\frac{1}{2}(\mathbf{v}^+ + \mathbf{v}^-)$	mean value at element face for a vector-valued or matrix-valued variable $\mathbf{v}$
$\underline{[\![\mathbf{v}]\!]}$	$\mathbf{v}^+\otimes\mathbf{n}^+ + \mathbf{v}^-\otimes\mathbf{n}^-$	jump value at element face for a vector-valued or tensor-valued variable $\mathbf{v}$

Flow field variables and derived quantities

Ω	open, bounded, and connected domain
d	space dimension
Γ	boundary of domain Ω
Γ_{in}	inflow boundary of domain Ω
Γ_{out}	outflow boundary of domain Ω
Γ_{wall}	wall boundary of domain Ω
$\mathbf{n}$	outer normal vector of Γ
ρ	density
$\mathbf{v}$	velocity vector
E	specific total energy
k	turbulent kinetic energy
ω	specific dissipation rate
$\tilde{\omega}$	logarithm of turbulence variable ω
$\tilde{\nu}$	model variable of the SA turbulence model
$\mathbf{u}$	vector of conservative variables, depending on context: $(\rho, \rho\mathbf{v}, \rho E)^\top$, $(\rho, \rho v_1, \ldots, \rho v_d, \rho E, \rho k, \rho\omega)^\top$, $(\rho, \rho\mathbf{v}, \rho E, \rho k, \rho\tilde{\omega})^\top$ or $(\rho, \rho\mathbf{v}, \rho E, \rho\tilde{\nu})^\top$
p	pressure
e	inner energy
$\mathcal{K}$	thermal conductivity coefficient
T	temperature
λ	bulk viscosity
μ	dynamic viscosity
$\mathcal{F}^c$	convective fluxes
$\mathcal{F}^v$	viscous fluxes
Re	Reynolds number
L	reference length of the body submerged in the fluid
$\mathcal{S}$	source terms
p_{eff}	effective pressure takes turbulence kinetic energy into account
μ_t	dynamic eddy viscosity
$\tilde{\omega}_l$	limitation of $\tilde{\omega}$
ν	kinematic eddy viscosity
P	production term in the $\tilde{\nu}$-equation
D	destruction term in the $\tilde{\nu}$-equation
d	wall distance
α	angle of attack
β	sideslip angle

$\mathbb{D}(\mathbf{v})$	$\frac{1}{2}(\nabla\mathbf{v} + (\nabla\mathbf{v})^\top)$	strain tensor
$\mathbf{f}_i^c$	$\mathcal{F}^c = (\mathbf{f}_1^c, \ldots, \mathbf{f}_d^c)$	columns of $\mathcal{F}^c$
$\mathbb{A}_i$	$\frac{\partial \mathbf{f}_s^c(\mathbf{u})}{\partial \mathbf{u}}$	Jacobi matrix of the flux $\mathbf{f}_i^c(\mathbf{u})$
λ_i		eigenvalue of the Jacobi matrix $\mathbb{A}_i$
$\mathbf{f}_i^v$	$\mathcal{F}^v = (\mathbf{f}_1^v, \ldots, \mathbf{f}_d^v)$	columns of $\mathcal{F}^v$
H	$E + \frac{p}{\rho}$	enthalpy
τ_{si}	$-\frac{2}{3}\mu\nabla\cdot\mathbf{v}\delta_{si} + 2\mu\mathbb{D}(\mathbf{v})_{si}$	viscous stress tensor
$\mathbb{G}_{sl}$		homogeneity tensor of the viscous fluxes
Ma	$\frac{\lvert\mathbf{v}\rvert}{a}$	Mach number
a	$\sqrt{\gamma\frac{p}{\rho}}$	speed of sound
$\tilde{\tau}_{si}$	$\tau_{si} + \frac{\mu_t}{\mu}\tau_{si}$	turbulent viscous stress tensor
τ_{si}^R	$2\mu_t\mathbb{D}(\mathbf{v})_{ij} - \frac{2}{3}(\mu_t\nabla\cdot\mathbf{v} + \rho k)\delta_{ij}$	Reynolds stress tensor in the source term for k
f_{v_1}	$\frac{\mathcal{X}^3}{\mathcal{X}^3+c_{v_1}^3}$	model function for μ_t in the SA turbulence model
$\mathcal{X}$	$\frac{\tilde{\nu}}{\nu}$	model function in the SA turbulence model
f_n		turbulence model function in the viscous flux for $\tilde{\nu}$
f_{t2}	$c_{t3}\exp\left(-c_{t4}\mathcal{X}^2\right)$	turbulence model function in source term for $\tilde{\nu}$
f_{v2}	$1 - \frac{\mathcal{X}}{1+\mathcal{X}f_{v1}}$	turbulence model function in source term for $\tilde{\nu}$
$\tilde{S}$	$\sqrt{2W_{ij}W_{ij}}$	turbulence model function in source term for $\tilde{\nu}$
W_{ij}	$\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right)$	turbulence model function in source term for $\tilde{\nu}$
c_{w1}	$\frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma}$	turbulence model function in source term for $\tilde{\nu}$
f_w	$g_1\left[\frac{1+c_{w3}^6}{g_1^6+c_{w3}^6}\right]^{1/6}$	turbulence model function in source term for $\tilde{\nu}$
g_1	$r_1 + c_{w2}(r_1^6 - r_1)$	turbulence model function in source term for $\tilde{\nu}$
r_1	$\min\left[\frac{\tilde{\nu}}{\hat{S}\kappa^2\mathsf{d}^2}, 10\right]$	turbulence model function in source term for $\tilde{\nu}$
S	$\frac{\tilde{\nu}}{\kappa^2\mathsf{d}^2}f_{v_2}$	turbulence model function in source term for $\tilde{\nu}$
$\hat{S}$		turbulence model function in source term for $\tilde{\nu}$
a_t	$\sqrt{\gamma\frac{p_{\text{eff}}}{\rho}}$	turbulent speed of sound
c_p		pressure coefficient
c_f		skin friction coefficient
C_L		lift coefficient
C_D		drag coefficient

Discretization notations

T_h	grid of the computational domain
κ	grid element
h	characteristic mesh size
p	polynomial degree
$L^2(\Omega)$	denotes the space of all measurable functions from Ω to $\mathbb{R}$
$C_0^\infty(\Omega)$	denotes the space of all functions which are arbitrarily often differentiable in Ω and equal to zero on Γ
α	multi-index
H^m	Sobolev space
$H^{m,b}$	broken Sobolev space
N	weak formulation of (2.18)
$\mathcal{V}_h^p$	space of vector-valued piecewise polynomial functions of degree p
$\mathcal{P}_p$	denotes the space of polynomial functions of degree at most p
f	interior face
$\mathcal{H}^{fl}$	numerical flux function
$\mathbf{G}$	matrix used in viscous flux
$\mathbf{G}^\top$	matrix used in viscous flux
$\underline{\delta}$	penalization term used in BR2 scheme
N_h	semi-linear form of the discontinuous Galerkin discretization
Λ	diagonal matrix of eigenvalues used in the convective flux
λ_i	eigenvalue used in the convective flux
T	eigenvector matrix used in the convective flux
$\mathbf{w}_i$	eigenvector used in the convective flux
$\mathcal{H}^{\text{Roe}}$	Roe flux
A	diagonal matrix used in the Roe flux
α_i	entries of A in the Roe flux
$\tilde{\omega}_{\text{wall}}$	values of $\tilde{\omega}$ ar Γ_{wall}
$d_{p,2D}$	basis dimension in 2D
$d_{p,3D}$	basis dimension in 3D

Algorithmic notations

l	level
$l_{\min}$	lowest level
$l_{\max}$	top level
$\mathbf{V}_l$	linear solution space on level l
$\mathbf{u}_l$	solution/state vector of the nonlinear problem on level l
$\mathbf{L}_l$	nonlinear operator on level l
$\mathbf{f}_l$	right handside of nonlinear problem on level l
$\hat{I}_l^{l-1}$	state vector restriction operator from level l to level $l-1$
I_l^{l-1}	restriction operator from level l to level $l-1$
$\hat{I}_{l-1}^{l}$	prolongation operator from level $l-1$ to level l
$\tilde{\mathbf{u}}_l$	return vector of the nonlinear multigrid algorithm
$\mathbf{m}_1$	array consiting of the numbers of pre-smoothing iterations per level
$\mathbf{m}_2$	array consiting of the numbers of post-smoothing iterations per level
τ	cycle type parameter for the multigrid algorithms
$\mathbf{R}_l$	nonlinear residual on level l
n	number of maximal iterations
s	residual tolerance in the Backward–Euler algorithm
$\underline{\mathsf{J}}$	fully implicit Jacobian matrix
$\underline{\mathsf{M}}$	the mass matrix
CFL	Courant-Friedrichs-Lewy number
β	exponent in the switched evolution relaxation method
Δt	pseudo time-step
ε	differencing parameter for finite differences
$\mathfrak{u}_l$	solution/state vector of the linear problem on level l
$\mathcal{L}_l$	linear operator on level l
$\mathfrak{f}_l$	right handside of linear problem on level l
r	linear residual
$\mathcal{K}$	Krylov subspaces
$\tilde{\mathfrak{u}}_l$	return vector of the linear multigrid algorithm
$\underline{\mathsf{L}}_l$	the sub-matrix of $\mathcal{L}_l$ extracted along lines
$\underline{\mathsf{N}}$	sub-matrix of $\mathcal{L}_l$ without line entries
r_{tol}	residual tolerance of the startup strategy
b_{it}	boolean parameter for the startup strategy
b_{sg}	boolean parameter for the startup strategy
s_{l-1}	number of elements in $\mathcal{T}_{l-1}$ on level $l-1$
$s_{l-1,m}$	number of fine elements to create the element $\kappa_{l-1,m}$ on level $l-1$
n_l	dimension of $\mathbf{V}_l$
$\mathbf{M}$	basis function scalar product matrix
$\mathbf{w}$	vector consisting of all basis coefficients to construct $\mathbf{u}$

a	used in the diffusion term of the advection-diffusion problem
b	flow solution used in the advection term of the advection-diffusion problem
$\mathbf{W}$	weight matrix for the line search algorithm
c_{rt}	coarsening rate
i_{GMRes}	number GMRes iterations
k_{NMG}	number of smoother steps in the nonlinear multigrid algorithm
i_{cycle}	number of linear multigrid cycles per preconditioning step
k_{LMG}	number of smoother steps in the linear multigrid algorithm

Constants

R		gas constant
c_p		isobaric heat capacity, specific heat at constant pressure
c_v		isochoric heat capacity, specific heat at constant volume
γ	$= 1.4$	ratio of heat capacities $\frac{c_p}{c_v}$
Pr	$= 0.72$	Prandtl number
Pr_t	$= 0.9$	turbulent Prandtl number
σ_k	$= 0.5$	turbulence model constant in the viscous flux for k
σ_ω	$= 0.5$	turbulence model constant in the viscous flux for ω
α_ω	$= 0.555$	turbulence model constant in P_ω
β_k	$= 0.09$	turbulence model constant in the source term for k
β_ω	$= 0.075$	turbulence model constant in the source term for ω
Tu	$= 0.001$	turbulence intensity, needed for the far field value of k
r	$= 0.01$	ratio of the far field turbulent viscosity and molecular viscosity
c_{v_1}	$= 7.1$	SA turbulence model constant in f_{v_1}
σ	$= 0.666$	turbulence model constant in the viscous flux for $\tilde{\nu}$
c_n	$= 16$	SA turbulence model constant in f_n
c_{b1}	$= 0.1355$	SA turbulence model constant in source term for $\tilde{\nu}$
c_{t3}	$= 1.2$	SA turbulence model constant in source term for $\tilde{\nu}$
c_{t4}	$= 0.5$	SA turbulence model constant in source term for $\tilde{\nu}$
c_{w2}	$= 0.3$	SA turbulence model constant in source term for $\tilde{\nu}$
c_{b2}	$= 0.622$	SA turbulence model constant in source term for $\tilde{\nu}$
c_{w3}	$= 2$	SA turbulence model constant in source term for $\tilde{\nu}$
κ	$= 0.41$	von Kármán constant in source term for $\tilde{\nu}$
δ_{ef}	$= 0.2$	Roe flux constant

List of Tables

4.1 Nonlinear V-cycle iteration cost within the current framework: turbulent L1T2 three element airfoil test case on an unstructured mesh. 57

4.2 Nonlinear iteration cost analysis on agglomerates. 62

4.3 Domain decomposition parallelization effect on current framework: turbulent L1T2 three element airfoil test case with a $p = 2$ computation on an unstructured mesh. 66

5.1 Test case mesh description. 69

5.2 Reference values for the lift and drag coefficients of the L1T2 test case. . . 69

5.3 Reference and DG_ON values for the lift and drag coefficients of the MDA 30P30N test case. 75

5.4 Reynolds-Averaged Navier-Stokes (RANS)-$k\omega$ values for the lift and drag coefficients of the MDA 30P30N test case. 77

5.5 L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: p- and h-startup strategy investigation. 96

5.6 MDA 30P30N $(p = 2)$ $k\omega$-computation on the 8 432 element mesh: p- and h-startup strategy investigation. 96

5.7 NASA Trap Wing $(p = 2)$ SA-computation on the 744 704 element mesh: p- and h-startup strategy investigation. 96

5.8 L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: startup tolerance investigation. 99

5.9 VFE-2 $(p = 2)$ $k\omega$-computation on a structured 13 816 element mesh: startup tolerance investigation. 99

5.10 L1T2 $(p = 3)$ SA-computation on the WUT mesh: startup tolerance investigation. 99

5.11 NASA Trap Wing $(p = 2)$ SA-computation on the 744 704 element mesh: startup tolerance investigation. 99

5.12 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 33 728 elements: influence of linear solver settings. 102
5.13 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 8 432 elements: influence of linear solver settings, see Figure 5.25. 103
5.14 L1T2 ($p = 2$) SA-computation on the unstructured WUT mesh: effect of coarse level linearization. 106
5.15 VFE-2 ($p = 2$) $k\omega$-computation on a structured mesh with 13 816 elements: effect of coarse level linearization. 106
5.16 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 33 728 elements: effect of coarse level linearization. 107
5.17 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 33 728 elements: effect of the linear multigrid preconditioner. 108
5.18 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 8 432 elements: effect of linear multigrid preconditioner. 110
5.19 MDA 30P30N ($p = 2$) SA-computation on a structured mesh with 33 728 elements: effect of linear multigrid preconditioner. 110
5.20 MDA 30P30N ($p = 2$) SA-computation on a structured mesh with 8 432 elements: comparison of the low level solver within the multigrid preconditioner. 112
5.21 MDA 30P30N ($p = 2$) SA-computation on a structured mesh with 33 728 elements: comparison of the low level solver within the multigrid preconditioner. 112
5.22 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 8 432 elements: comparison of the low level solver within the multigrid preconditioner. 113
5.23 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 33 728 elements: comparison of the low level solver within the multigrid preconditioner. 113
5.24 VFE-2 ($p = 2$) $k\omega$-computation on a structured mesh with 13 816 elements: comparison of the low level solver within the multigrid preconditioner. 113
5.25 L1T2 ($p = 3$) SA-computation on the unstructured CEN mesh: effect of the linear multigrid preconditioner. 114
5.26 MDA 30P30N ($p = 2$) SA-computation on a structured mesh with 8 432 elements: influence of different linear solver algorithms. 117
5.27 MDA 30P30N ($p = 2$) $k\omega$-computation on a structured mesh with 8 432 elements: influence of different linear solver algorithms. 117

5.28 VFE-2 $(p = 2)$ $k\omega$-computation on a structured mesh with 13 816 elements: effect of coarse level linearization. 118
5.29 L1T2 $(p = 2)$ $k\omega$-computation on the unstructured CEN mesh: nonlinear multigrid cycle type investigation. 120
5.30 L1T2 $(p = 2)$ SA-computation on the unstructured CEN mesh: nonlinear multigrid cycle type investigation. 120
5.31 MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 33 728 elements: nonlinear multigrid cycle type investigation. 121
5.32 MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 134 912 elements: nonlinear multigrid cycle type investigation. 121
5.33 VFE-2 $(p = 2)$ $k\omega$-computation on a structured mesh with 110 528 elements: nonlinear multigrid cycle type investigation. 121
5.34 L1T2 $(p = 3)$ SA-computation on the unstructured CEN mesh: nonlinear multigrid cycle type investigation. 121
5.35 L1T2 $(p = 2)$ $k\omega$-computation on the unstructured CEN mesh: effect of the number of coarse level smoother steps within the nonlinear multigrid algorithm. 123
5.36 MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 134 912 elements: effect of the number of coarse level smoother steps within the nonlinear multigrid algorithm. 123
5.37 L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: solver comparison with same settings, see Figure 5.32. 127
5.38 L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: higher CFL numbers in comparison to Table 5.37. 127
5.39 L1T2 $(p = 3)$ SA-computation on the CEN mesh: solver comparison with same settings. 128
5.40 MDA 30P30N $(p = 2)$ $k\omega$-computation on the 33 728 element mesh: solver comparison with same settings. 129
5.41 MDA 30P30N $(p = 2)$ SA-computation on the 134 912 element mesh: solver comparison with same settings. 129
5.42 VFE-2 $(p = 2)$ $k\omega$-computation on the 110 528 element mesh: solver comparison with same settings. 129
5.43 NASA Trap Wing $(p = 2)$ SA-computation on the 744 704 element mesh: solver comparison with same settings. 129

List of Figures

3.1 Unstructured higher order L1T2 mesh created via GMSH. 34

3.2 Structured higher order MDA 30P30N mesh created via coarsening and additional points: full geometry (left), slate zoom (right). 35

4.1 Overview of the structure of Chapters 2-4. 37

4.2 V-cycle with four levels (left), W-cycle with four levels (right). 40

4.3 Sawtooth-cycle with four levels. 45

4.4 Computed lines (red) in a mesh of an airfoil, created with a scalar advection-diffusion problem 53

4.5 Jacobian breakdown for a quadrangle 2D mesh. 54

4.6 Unstructured L1T2 three element airfoil mesh with 23 824 elements and two agglomerations. 56

4.7 Overview of the solver algorithms implemented in the framework. 60

5.1 Mach number distribution on the L1T2 slat of a third order RANS-$k\omega$ solution on an unstructured 23 824 element mesh. 68

5.2 Top: Unstructured L1T2 mesh with 25 757 elements (WUT mesh), Bottom: Unstructured L1T2 mesh with 23 824 elements (CEN mesh). 69

5.3 Mesh convergence of error in lift (left) and drag coefficients (right) for reference computations and DG results. 70

5.4 Comparison between finite volume (FV) RANS-SA reference solution (black) and results for the RANS-SA equations for $p = 2$ (blue), $p = 3$ (green) and $p = 4$ (red) on the CEN mesh in terms of computed pressure coefficient (left) and skin friction coefficient (right). Symbols indicate experimental data. 71

5.5 Results for the RANS-$k\omega$ equations for $p = 2$ (blue) on the CEN mesh in terms of computed pressure coefficient (left) and skin friction coefficient (right). Symbols indicate experimental data. 72

5.6 Comparison between FV RANS-SA reference solution (black) and results for the RANS-SA equations $p = 2$ (blue) and $p = 3$ (green) on the WUT mesh in terms of computed pressure coefficient (left) and skin friction coefficient (right). Symbols indicate experimental data. 72

5.7 Nonlinear convergence of all residual components for a L1T2 $k\omega$-computation at $p = 2$ with a h-startup strategy on the CEN mesh. 73

5.8 Pressure plot of a $p = 2$ solution on a structured mesh with 33 728 elements. 74

5.9 Top: Lift of RANS-$k\omega$ eq. in correlation to the mesh size. Bottom: Lift of RANS-SA eq. in correlation to the mesh size. 75

5.10 Top: Drag of RANS-$k\omega$ eq. in correlation to the mesh size. Bottom: Drag of RANS-SA eq. in correlation to the mesh size. 76

5.11 Nonlinear convergence of all residual components for an MDA 30P30N SA-computation with $p = 2$ on the 33 728 element mesh. 77

5.12 Pressure contours on the wing and Mach number on the symmetry plane and streamline plot of a $p = 2$ solution on the mesh with 110 528 elements and experimental data represented in the slice of vorticity. 78

5.13 Subsonic flow around the VFE-2 delta wing: Convergence of lift (top) and drag (bottom) in correlation to the mesh size. 79

5.14 Comparison of c_p distribution for a $p = 2$ RANS-kw result on the mesh with 884 224 elements and experimental data. Left: On the upper wing with experimental data on the left and flow computation data on the right. Right: Slice at 80% chord of the upper wing. 80

5.15 Nonlinear convergence of all residual components for an VFE-2 $k\omega$-computation at $p = 2$ on the 110 528 element mesh. 81

5.16 Mach number distribution plotted on the symmetry plane and the c_p distribution on the wall. Left: $p = 3$ SA-results on the coarse mesh with 93 088 elements. Right: $p = 2$ SA-results on the fine mesh with 744 704 elements. 82

5.17 Nonlinear convergence of all residual components for the NASA Trap Wing RANS-SA computations for $p = 2$ on the 93 088 element mesh (top) and on the fine 744 704 element mesh (bottom). 83

5.18 Convergence of lift (left) and drag (right) coefficient for high order computations in comparison to experimental data. 84

5.19 Comparison of the c_p distribution at different slices on the wing for $p = 3$ SA-results on the coarse mesh with 93 088 elements (left), $p = 2$ SA-results on the fine mesh with 744 704 elements (right) and experimental data ■. 85

5.20 Nonlinear convergence for a L1T2 $k\omega$-computation at $p = 2$ on the CEN mesh for different combinations of solver algorithms. 87

5.21 Nonlinear convergence of all residual components for $p = 2$ computation of the L1T2 on the CEN mesh for the RANS-$k\omega$ equations (left) and the RANS-SA equations (right) with no startup strategy applied. 93

5.22 Nonlinear convergence of all residual components for $p = 2$ computation of the L1T2 on the CEN mesh for the RANS-$k\omega$ equations (left) and the RANS-SA equations (right) with startup strategy applied. 94

5.23 Nonlinear convergence for a L1T2 SA-computation at $p = 3$ on the WUT mesh for different h-startup strategy tolerances. 97

5.24 Critical solver elements for trade-off between nonlinear and linear iterations investigation are highlighted in red within the framework overview. . 101

5.25 MDA 30P30N $p = 2$ $k\omega$-computation, from Table 5.13, on a structured mesh with 8 432 elements with a 10^{-6}-$p(2)$-startup strategy. 103

5.26 Critical solver elements for the coarse level linearization investigation are highlighted in red within the framework overview. 105

5.27 Critical solver elements for the investigation of the GMRes preconditioner are highlighted in red within the framework overview. 107

5.28 MDA 30P30N $p = 2$ $k\omega$-computation on a structured mesh with 33 728 elements, from Table 5.17 (rows 2,4,5): The top level of a 10^{-6}-$h(4)$-startup strategy. 109

5.29 Critical solver elements for the linear solver investigation are highlighted in red within the framework overview. 116

5.30 Critical solver elements for the investigation of the nonlinear multigrid algorithm are highlighted in red within the framework overview. 119

5.31 L1T2 $(p = 3)$ SA-computation on the CEN mesh, from Table 5.34: number of top level Backward–Euler iterations. 124

5.32 Nonlinear convergence for a L1T2 $k\omega$-computation at $p = 2$ on the CEN mesh for different combinations of solver algorithms with an h-startup strategy. 125

5.33 Comparison between RANS-SA DG_ON results for $p = 2$ (blue), $p = 3$ (green) and $p = 4$ (red) on the CEN mesh in terms of the convergence history of the density residual component on the left and the error in the lift coefficient in comparison to reference computations on the right. . . . 132

5.34 Convergence rates of RANS-SA computations of DG_ON result for $p = 2$ on the MDA 30P30N mesh hierarchy. 133

5.35 Mesh convergence of the error in lift over DoF (left) and work units (right) for reference computations and DG results of the L1T2 test case. 136
5.36 Mesh convergence of error in drag over DoF (left) and work units (right) for reference computations and DG results of the L1T2 test case. 136
5.37 Results of the MDA 30P30N testcase in combination with the RANS-SA equations. Left: Lift and drag in correlation to work units. Right: Lift and drag in correlation to DoF. 137
5.38 Results of the MDA 30P30N testcase in combination with the RANS-$k\omega$ equations. Left: Lift and drag in correlation to work units. Right: Lift and drag in correlation to DoF. 138
5.39 Mesh and order convergence of the error in the lift coefficient over DoF (left) and work units (right) of the NASA Trap Wing test case. 139

List of Algorithms

1 Nonlinear Multigrid Algorithm $NMG(\mathbf{f_l}, \mathbf{u}_l, \mathbf{m}_1, \mathbf{m}_2, l, l_{\text{min}}, \tau)$ 39
2 Backward–Euler $BWE_l(\mathbf{u}_l, n, s)$ 41
3 Linear Multigrid Algorithm $LMG(\mathfrak{f}_l, \mathfrak{u}_l, \mathbf{m}_1, \mathbf{m}_2, l, l_{\text{min}}, \tau)$ 44
4 Line–Jacobi method $LJ_l(\mathbf{u}_{l,k,0}, n)$ 46
5 Startup strategy $Startup(l_{\text{min}}, l_{\text{max}}, r_{\text{tol}}, b_{\text{sg}}, b_{\text{it}})$ 48
6 Line Search Algorithm $LSA(\mathbf{W})$ 54

1 Introduction

Computational Fluid Dynamics (CFD) advanced substantially in the past decades [1]. Moreover, CFD tools became essential in the design process and analysis of modern aircraft. In addition to wind-tunnel experiments, CFD has the ability to simulate a large range of flow conditions for all relevant target values in the entire flow field around an aircraft. By the enhancement of numerical methods and computer systems, the complexity of problems solved has increased drastically since the first two-dimensional potential flow simulations. Higher-order methods are an important contribution to today's CFD, because they have the potential to reduce the computational cost required to obtain the desired accuracy in CFD simulations. The most common CFD methods today are second-order finite volume (FV) methods [31]. For these methods the error scales as $||u_h - u|| \sim h^2$, where u_h is the numerical solution, u the analytical solution and h a characteristic mesh size. In contrast to that, the error of higher-order methods scales like $||u_h - u|| \sim h^k$, with $k > 2$, if the underlying solution is sufficiently smooth.

In this work a higher-order discontinuous Galerkin (DG) method is used. DG methods were introduced in the 1970s by Reed and Hill to solve the hyperbolic neutron transport equation [58]. Furthermore, they are finite element (FE) methods with the specific characteristic that the ansatz and test functions are discontinuous at shared boundaries between elements and the element to element coupling exists only through the flux at these boundaries. This FE method enables an intuitive and convenient treatment of convective terms via approximate Riemann solvers employed as numerical flux functions over element interfaces like in FV methods. Since DG methods are FE methods, polynomials are defined on every element in the computational mesh. These polynomials build a basis for the solution space of the underlying discretization. Hence, DG methods can be characterized in two ways. On the one hand, DG methods could be seen as continuous standard FE methods on each mesh element but allowing discontinuities over faces between elements. On the other hand, DG methods could be seen as FV methods with a possible non-constant function value on each mesh element, e. g. a polynomial function. Thus, for constant values on each mesh element DG methods can be stated as a first order FV method. With that in mind, DG methods combine features from FE methods and FV methods. Another reason to choose DG methods is that the usage of unstructured grids

is very simple due to the local formulation of DG methods. This issue makes the DG methods very suitable for grid adaptation. Moreover, for DG methods all characteristics of FE methods are sustained, e.g. an error estimation can easily be incorporated as well. Some usage of DG methods in CFD is shown in [4, 5, 13, 14, 22, 26, 35, 37, 52, 64, 68, 73–76].

In CFD, the analysis of turbulent flows employing steady-state computations based on Reynolds-Averaged Navier-Stokes (RANS) equations in combination with a turbulence model are still state of the art for many applications in exterior aerodynamics. These equations are derived from the Navier-Stokes equations, which describe the compressible, viscous flow of a Newtonian fluid. This system of equations consists of the transport equations for mass, momentum and total energy. Derivations can be found in many standard textbooks, e. g. [11]. This thesis will focus on a turbulent flow around a body, e.g. an airfoil, based on the mentioned equations. From the mathematical point of view this system of nonlinear partial differential equations has an hyperbolic-elliptic nature. Without simplifications and limitations no analytical solution of these partial differential equations is known [20]. The numerical solutions for these equations comprise many different physical phenomena, for instance the solution may contain boundary layers, shocks, vortices and separation. Solving such a system of hyperbolic-elliptic nonlinear partial differential equations numerically, e.g. the RANS equations in combination with a turbulence model, is not an easy task. DG results are relatively rare for this particular application [5, 13, 14, 26, 35, 37, 52, 64, 76].

One of the reasons for this might be that highly stretched meshes typically used for an optimal resolution of turbulent boundary layers at high Reynolds number flows, and source terms present in the turbulence models contribute to an increased stiffness of the resulting algebraic system of equations that has to be solved. Designing efficient iterative solvers is thus a difficult but important task. In order to solve the stationary RANS equations in combination with a turbulence model several authors suggested implicit time-stepping schemes that are close to Newton's method. The idea of these methods is to consider for the stationary RANS equations unsteady flow equations and to advance the flow in time until a stationary state is reached.

Such a method is the linearized Backward-Euler approach which can be characterized as a one-stage Rosenbrock method [60] and is predominantly used for recent results [5, 14, 26, 64]. However, these implicit solver are very dependent on a good initial guess of the desired flow field. Especially for fine meshes and higher polynomial degrees these solvers tend to diminish their performance in terms of CPU time and memory consumption.

Therefore, this thesis investigates multilevel solver algorithms and suitable startup strategies to enhance the solution process in terms of overall CPU time and robustness. In addition to that the proposed solvers should be memory efficient and should not contain

bottle necks concerning parallelization. These factors are also considered in the design process of the proposed solver algorithms in this work.

1.1 Literature review and contribution to the field

As mentioned above, the main focus lies on introducing solver algorithm for DG discretizations of steady-state turbulent flows described by the steady-state RANS equations in combination with a turbulence model.

The Newton-like methods, e. g. the linearized Backward-Euler algorithm, in combination with preconditioned Krylov subspace methods as linear solvers are nearly black box approaches for the nonlinear equations handled in this work. In contrast to that, solver algorithms which utilize additional knowledge about the equations, which are going to be solved, are employed here. Considered are both level sequencing and linear as well as nonlinear multigrid variations [70]. Based on either lower order discretizations or agglomerated coarse meshes, the resulting algorithms can be characterized as either p- or h-multigrid, respectively.

In the following, a brief overview is given about solver algorithms applied in CFD. A implicit method is often suggested as a solver algorithm in the DG community to solve the steady-state RANS equations, e.g. [5, 14, 26, 76]. It is well-known that these implicit methods work best at the end of the iterative procedure in the regime of asymptotic convergence, i.e., when the iterative solution is already close to the converged solution in some sense. The dependency of the algorithm on a good initial guess increases with more difficult problems, in particular with finer meshes or increasing order of accuracy of the discretization method. Employing mesh sequencing for structured meshes and order sequencing helps to alleviate this problem and can be seen as a standard approach in this field. Thus, using high-order on coarse meshes is beneficial for this solution strategy. The same is true for mesh-adaptive computations starting on coarse initial meshes.

In contrast to multi-block structured meshes, which incorporate multiple resolution levels in most case, unstructured meshes have to be generated at the target mesh resolution and typically do not incorporate a nested sequence of coarser meshes. Still, a higher extent of automation motivates the use of unstructured meshes for complex geometries. The DG approach is immediately capable of achieving high order on this type of mesh. With most available meshing tools it is rather difficult to generate meshes that are very coarse globally but still feature a reasonable resolution of both geometric and flow features. Thus, unstructured meshes tend to be finer than the coarsest structured ones. Due to the increased mesh size, they are usually treated with moderately high order in the DG context. Recent results indicate that in this area of application the most gains in

terms of computational effort per accuracy are gained by approaches up to third or fourth order [77]. In this range a beneficial balance is found for the computational cost per degrees of freedom (DoF) between DoF gained by order enhancement or mesh resolution to resolve turbulent flows. The only option to enhance the solution process in terms of mesh sequencing for a unstructured fine mesh is the agglomeration of the given unstructured mesh, which was introduced in a DG context in [3, 4]. Coarse meshes obtained by agglomeration of unstructured meshes consist of general polyhedra. Therefore, it is mandatory to use basis functions for the DG discretization which are formulated in the physical space and not with the aid of reference elements [45]. From this point, it seems a reasonable approach to use the resulting agglomerations not only for employing mesh sequencing but also for agglomeration h-multigrid algorithms. However, such an approach has not yet been published by other authors in literature for DG methods for turbulent flows. Only for the Euler equations and a simple NACA airfoil flow case a nonlinear agglomeration h-multigrid method using V-cycles is described in [68]. More investigations, e. g. other test cases, system of equations, cycle types, meshes or a linear h-multigrid are not shown in [68] but will be considered in detail in this thesis. Moreover, results from a nonlinear h-multigrid algorithm for non-turbulent flows were previously presented for a second order DG discretization on structured grid hierarchies [32]. The nonlinear and linear h-multigrid algorithm presented in this thesis are intended for applications featuring turbulent flows and are formulated on unstructured grids and their agglomerations with arbitrary polynomial degree p.

For a DG discretization, a p-multigrid algorithm is particularly attractive, as it does not include additional meshes and can thus be realized with minimal additional implementational effort. Therefore, this approach was suggested as a solver algorithm for inviscid and laminar viscous flows by many authors [7, 22, 23, 44, 46, 56, 65]. However, results indicate that in the context of turbulent flows the nonlinear p-multigrid algorithm is not always a stable algorithm, yet [22, 76]. The lack of publications in this application area might indicate that other authors share this experience. Here, a nonlinear p-multigrid algorithm is investigated further, with additional alteration, e.g. using a Galerkin transfer for the Jacobian on the lower levels. In addition, a linear p-multigrid algorithm used as a preconditioner [14] or linear solver is introduced as well as combinations of nonlinear and linear p-multigrid algorithms. The coarsest level will always be $p = 1$ as seen in [65, 76], but in contrast to [14]. This avoids the inconsistent first order discretization ($p = 0$) in the presence of viscous fluxes and source terms. Such a combination of linear and nonlinear multigrids for turbulent DG discretizations has not been shown before, neither in a p- nor for a h-multigrid context. This combination of either h- or p-multigrid algorithms between a nonlinear and linear multigrid for a turbulent DG discretization with an arbitrary polynomial degree p is introduced in this work and it gives satisfactory results in robustness and efficiency.

Newton-like methods often introduce very high memory requirements, as the full Jacobian of a high order discretization has to be stored. The necessity to use efficient preconditioners for iterative solvers of the linear systems often yields a further increase in memory consumption. One of the points why DG is not able to calculate industrial near test cases yet, is the unnecessary high memory consumption for the solvers mainly used up to now. In most cases, these Newton-like methods could be realized in a matrix-free fashion and, therefore, the high memory consumption could be dealt with if the preconditioner could be implemented in a matrix-free fashion as well. Unfortunately, most of the time a preconditioner is used which has very high memory consumption as well [5, 26]. Other authors addressed the issue of high memory consumption by the preconditioner applying element and line implicit methods in a DG context in combination with a serial implementation of a linear p-multigrid preconditioner [14]. On this basis, line implicit methods as stand-alone algorithms or in combination with a linear p- or h-multigrid algorithm applied as a linear solver or preconditioner is addressed to tackle this issue. Moreover, the implementation in this thesis works highly parallel, with a domain decomposition parallelization.

In this thesis no combinations of h- and p-multigrid algorithms, so called hp-multigrids algorithms, are investigated. The absence of computational results for such multigrid combinations in DG method for turbulent flows indicate a lack of solution strategies for these cases. Proposed algorithms which use an h-multigrid algorithm on one or more levels of a p-multigrid algorithm for non-turbulent flows can be seen in [51, 65, 71, 72, 80]. The results shown in this work indicate that a sole $p-$ or h-multigrid approach is superior to a single grid computation in terms of CPU time. It is debatable of a hp-multigrid combination can decrease the CPU time further in comparison to a sole $p-$ or h-multigrid approach because the additional work has to be absorbed by a superior convergence histroy in order to save CPU time.

Employed for the nonlinear multigrid algorithms as underlying relaxation scheme is a linearized Backward-Euler scheme based on local pseudo-time steps that can be considered as a stabilized Newton's method and is also used predominantly as single-level solver [5, 14, 26, 64]. Presented are numerical examples in order to analyze the performance of the suggested algorithms with respect to both algorithmic convergence properties and run-time behavior in comparison with a single level Backward-Euler scheme, applied on the sequence of levels resulting from the multigrid algorithm. A similar comparison between h-multigrid and single grid solver is also considered in the context of a nonlinear FV h-multigrid solver for turbulent flows [30]. Various other investigations are presented including preconditioning investigations and several multigrid alterations.

To conclude, the proposed framework includes linear and nonlinear p- and h-multigrid algorithms and single level solvers combined in one CFD framework tackling the RANS equations in combination with a turbulence model discretized with a DG method. The

parallelization is realized via a full domain-decomposition of all data. All h-multigrid algorithms are based on agglomerations and results are shown up to fifth order. Moreover, two turbulence models are considered to validate the proposed solver algorithms for different DG discretizations of turbulent flows. Since this framework is very versatile various and new combinations of solver algorithms and startup strategies are discussed.

1.2 Objective

The main focus lies on the development of robust, efficient and scalable solver algorithms for a higher-order DG discretization of steady-state turbulent flow problems in aerodynamic applications. Focus is put on nonlinear and linear p- and h-multigrid methods in various combinations. The computation of reliable numerical results for flow problems in two and three dimensions faster in CPU time in comparison to a state of the art single level solver is a desirable result.

The nonlinear and linear h-multigrid algorithms presented in this thesis are intended for turbulent flows and are formulated on unstructured grids and their agglomerations with arbitrary polynomial degree p. Such approaches have not yet been investigated and published by other authors in literature for DG methods in combination with turbulent flows. The combination of either h- or p-multigrid algorithms between a nonlinear and linear multigrid for a turbulent DG discretization with an arbitrary polynomial degree p, which has also not been demonstrated before, is introduced in this work and gives satisfactory results in terms of robustness and efficiency.

1.3 Thesis outline

A brief introduction of the governing equations applied in this thesis is given in Chapter 2. Starting with the time-dependent compressible Navier-Stokes equations a steady-state formulation is introduced. These equations are then stated in a flux formulation and the RANS equations are briefly introduced in combination with two different turbulence models: a one equation model, namely the negative Spalart–Allmaras turbulence model [57, 66], and a two equation turbulence model, namely the Wilcox $k\omega$ turbulence model [78, 79], cf. Section 2.3.

In Chapter 3 an introduction to a DG discretization of the RANS equations for both turbulence models is given. In particular boundary conditions, numerical flux functions, basis functions and mesh related issues are discussed.

The only difference between the p- and h-multigrid algorithms in this thesis is the use of different coarse level DG discretizations and, therefore, transfer operators, cf. Section 4.6. All other ingredients like smoothers, timestep control, usage of a Galerkin transfer, startup strategy, etc. stay the same for all kinds of multigrid algorithms. This general formulation of the multigrid methods are presented in Chapter 4, along with the few distinctions that are still necessary. Also a Backward-Euler scheme in combination with a Krylov subspace methods is shown, cf. Section 4.2. The last section in this chapter provides an overview of the implemented CFD framework. Furthermore, parallelization and a matrix-free implementation is addressed.

All numerical investigations can be found in Chapter 5, which is structured in the following way. At first, three numerical test cases are introduced in Section 5.1. A validation against data from other CFD solvers and experimental data, if available, is presented for each test case. Moreover, all meshes for each test case are introduced and described. Hence, Section 5.1 only addresses the fact that the flow data from the proposed framework is trustworthy and shows similar results as data from other CFD codes. All numerical investigations within the proposed framework are presented in Section 5.2. Here, the test cases from the previous section are used and one of the main goals of Section 5.2 is that for the investigated phenomena a test case independence can be shown. Only in very few points a distinction has to be made. In general, all statements and conclusions drawn from Section 5.2 are true for all presented test cases and meshes. At last, all test cases from Section 5.1 are revisited but now a "best practice"-solver from the current framework is compared against other CFD codes in terms of CPU time and accuracy. In addition to that, a last challenging test case application for DG, the NASA Trap Wing, is introduced and results from the current framework in comparison to other CFD codes is shown.

Conclusions and an outlook are stated in Chapter 6.

2 Governing equations in CFD

In this chapter the compressible Navier-Stokes equations are introduced. These equations describe a continuum flow of a viscous, compressible fluid. This system of nonlinear partial differential equations (PDEs) consists of conservation equations for mass, momentum and total energy. A derivation of the equations can be found in, e. g. Blazek [11]. In most cases it is impractical to solve a numerical representation of this system of equations directly because direct numerical simulations of the Navier-Stokes equations are extremely costly. This is due to the large range of flow scales from the magnitude of the simulation region, e.g. an aircraft, to the size of local eddy structures.

To overcome this problem a mass-weighted time averaging introduced by Reynolds can be performed [16]. This approach leads to the Reynolds-averaged Navier-Stokes (RANS) equations. One of the terms resulting from the averaging, namely the Reynolds stress tensor, can be handeld via the Boussinesq approximation [11]. This ansatz connects the Reynolds stresses with the velocity gradient. The resulting proportionality factor, namely the eddy viscosity, has to be handled via a turbulence model [16].

After introducing the instationary Navier-Stokes equations in Section 2.1, Section 2.2 gives the flux formulation of the steady-state Navier-Stokes equations and Section 2.3 the steady-state Reynolds-Averaged Navier-Stokes (RANS) equations. Here, two different turbulence models are considered, the two equation Wilcox $k\omega$ turbulence model [78, 79] and the one equation turbulence model of Spalart–Allmaras (SA) [66]. Both turbulence models have been wildly used in the DG community with some specific modifications which also will be described in this section.

2.1 The compressible Navier-Stokes equations

Let $\Omega \subset \mathbb{R}^d$ be an open, bounded and connected domain in d dimensions, $d = 2, 3$, with the boundary denoted by $\Gamma := \partial\Omega$. Furthermore let Ω be a Lipschitz domain [17]. Then the Navier-Stokes equations for compressible linear Newtonian fluid [11] are stated as a parabolic system of nonlinear partial differential equations on the domain Ω in a time interval $(0, t_{end})$ with $0 < t_{end} < +\infty$ and conservative variables $\mathbf{u} = (\rho, \rho\mathbf{v}, \rho E)^\top$ with $d+2$

components [56]. These compressible Navier-Stokes equations consist of conservation equations for mass, momentum and energy, and are given by:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) &= 0, \\ \frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v}) &= -\nabla p + \nabla \cdot (2\mu \mathbb{D}(\mathbf{v})) + \nabla(\lambda \nabla \cdot \mathbf{v}), \\ \frac{\partial \rho E}{\partial t} + \nabla \cdot ((\rho E + p)\mathbf{v}) &= \nabla \cdot (\lambda \mathbf{v} \nabla \cdot \mathbf{v}) + \nabla \cdot (2\mu \mathbb{D}(\mathbf{v})\mathbf{v}) + \nabla \cdot (\mathcal{K} \nabla T) \text{ in } \Omega, \end{aligned} \tag{2.1}$$

where $\mathbf{u} : [\Omega \times (0, t_{end})]^{d+2} \to \mathbb{R}^{d+2}$ and initial values are given by

$$\mathbf{v}(\mathbf{x}, 0) = \mathbf{v}^0(\mathbf{x}),\ \rho(\mathbf{x}, 0) = \rho^0(\mathbf{x}),\ T(\mathbf{x}, 0) = T^0(\mathbf{x}),\ \mathbf{x} \in \Omega. \tag{2.2}$$

Here ρ denotes the density, $\mathbf{v} := (v_1, \ldots, v_d)^\top$ the velocity vector and p the pressure. The specific total energy E is given by $E = e + \frac{1}{2}\|\mathbf{v}\|_2^2$, where e is the internal energy. Furthermore $\mathcal{K}$ is the thermal conductivity coefficient, T the temperature and λ, μ are the viscosity coefficients. For Newtonian fluids the following assumption for the viscosity coefficients λ, μ holds

$$\lambda + \frac{2}{3}\mu = 0. \tag{2.3}$$

This property, which is called the bulk viscosity, is responsible for the energy dissipation in a fluid of uniform temperature during a change in volume at finite rate [11]. Hereby, the equations (2.1) can be stated based on the single viscosity coefficient μ only. $\mathbb{D}(\mathbf{v})$ is defined as $\mathbb{D}(\mathbf{v}) := \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^\top)$. The operator $\nabla \mathbf{v} \in \mathbb{R}^{d \times d}$ is defined as $\{\frac{\partial v_i}{\partial x_j}\}_{(i,j) \in \underline{d} \times \underline{d}}$. Here $\underline{d}$ denotes the set $\{1, \ldots, d\}$. The tensor $(\rho \mathbf{v} \otimes \mathbf{v})$ is defined as $\{(\rho v)_i v_j\}_{(i,j) \in \underline{d} \times \underline{d}}$ and the divergence $\nabla \cdot \mathbb{D}(\mathbf{v})$:

$$(\nabla \cdot \mathbb{D}(\mathbf{v}))_i := \sum_{j=1}^{d} \frac{\partial \mathbb{D}_{ij}}{\partial x_j}, i \in \underline{d}, \tag{2.4}$$

These equations formulated in (2.1) are to understand in a weak formulation which is introduced in a later stage of this work.

As the compressible Navier-Stokes equations (2.1) contain two more variables than number of equations, i. e. seven unknown flow field variables namely $\rho, v_1, v_2, v_3, E, p$ and T for $d = 3$, they must be closed using material properties , i. e. two additional equations have to be supplied. In particular, thermodynamic relations are required between the state variables, for example the pressure as a function of density and temperature, and the internal energy or the enthalpy as a function of pressure and temperature. To this end, the thermodynamic pressure p is assumed to be given by the equation of the state of a ideal gas $p = \rho R T$. Here T denotes the temperature and the gas constant $R = c_p - c_v$ is given by the difference between the specific heat capacities at constant pressure c_p and

at constant volume c_v. The ratio of c_p and c_v is assumed to be constant, $\gamma = \frac{c_p}{c_v}$. Also the following energy equation holds, $E = c_v T + \frac{1}{2}\|\mathbf{v}\|_2^2$. Hence, the temperature is given by $T = \frac{1}{c_v}(\frac{1}{2}\|\mathbf{v}\|_2^2 - E)$. With that in mind the thermodynamic pressure formula can be rewritten, with the equation of state of an ideal gas, as

$$p = (\gamma - 1)\left(\rho E - \frac{\rho}{2}\|\mathbf{v}\|_2^2\right) = \rho(\gamma - 1)e. \tag{2.5}$$

Furthermore, with the help of Fourier's law, which describes the heat flux, the internal energy or the enthalpy can be written as a function of pressure and temperature. $\mathrm{Pr} := \frac{\gamma \mu c_v}{\mathcal{K}}$ is defined to be the characteristic Prandtl number [55], which is a dimensionless number and results from the ratio between the momentum diffusivity and the heat transfer eddy diffusivity. Then an equation for the thermal conductivity coefficient can be written as:

$$\mathcal{K} = \frac{\mu c_p}{\mathrm{Pr}}. \tag{2.6}$$

For dry air the constants are given by: $\gamma = 1.4$ and $\mathrm{Pr} = 0.72$ [55]. This closes the set of equations for the given variables $\rho, \mathbf{v}, E, p$ and T.

At the end of this section the model problem of the Navier-Stokes equations is formulated as follows:

Find a function $\mathbf{u} = \mathbf{u}(x, t)$ which solves the Navier-Stokes equations (2.1) with given initial values (2.2) and the boundary conditions described in [11, 56].

The existence of a strong solution for this problem is in most instances unsolved. Analytic solutions are known for simplifications of this problem only, some of them are shown in [20, 53].

2.2 Flux formulation of the steady-state Navier-Stokes equations

For many flight conditions a steady-state solution can be assumed, hence, the results from steady-state computations are still one of the backbones in CFD. Although no analytic solutions are available windtunnel experiments indicate that these steady flow fields exist. These flow conditions are the main focus of interest in this work, therefore all numerical experiments treat the solution of the steady-state equations.

Therefore in this section, the steady-state Navier-Stokes equations in a flux formulation are presented. Note that, less computational effort is needed to solve a numerical discretization of the steady-state equations in comparison to the time-dependent formulation. In addition,

The steady-state compressible Navier-Stokes equations can be rewritten in conservative form based on convective and viscous fluxes $\mathcal{F}^c$ and $\mathcal{F}^v$, corresponding to first and second order derivatives, respectively, as follows [78, 79],

$$\nabla \cdot (\mathcal{F}^c(\mathbf{u}) - \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u})) = 0 \quad \text{in } \Omega, \tag{2.7}$$

where $\mathbf{u}$ is the vector of the conservative variables $\mathbf{u} = (\rho, \rho v_1, \ldots, \rho v_d, \rho E)^\top$. Here, the divergence operates on the d variables $\mathbf{f}$ of $\mathcal{F}^c$ and $\mathcal{F}^v$ as in (2.4).

The convective flux is given by $\mathcal{F}^c(\mathbf{u}) = (\mathbf{f}_1^c, \ldots, \mathbf{f}_d^c) \in \mathbb{R}^{(d+2)\times d}$ and

$$\mathbf{f}_s^c(\mathbf{u}) = \mathbf{u}v_s + (0, \delta_{1s}, \ldots, \delta_{ds}, v_s)^\top p. \tag{2.8}$$

Here, δ_{ij} denotes the Kronecker delta symbol, with $\delta_{ij} = 1$ for $i = j$ and $\delta_{ij} = 0$ else. Furthermore, $\mathbb{A}_s(\mathbf{u}) \in \mathbb{R}^{(d+2)\times(d+2)}$ denotes the Jacobi matrix of the flux $\mathbf{f}_s^c(\mathbf{u})$, i.e.,

$$\mathbb{A}_s(\mathbf{u}) := \frac{\partial \mathbf{f}_s^c(\mathbf{u})}{\partial \mathbf{u}},\ s \in \underline{d}, \tag{2.9}$$

thus $(\mathbb{A}_s(\mathbf{u}))_{ij} = \partial_j(\mathbf{f}_s^c(\mathbf{u}))_i\ ,\ i, j \in \underline{d+2}$. In the two- and three-dimensional cases, the flux Jacobians $\mathbb{A}_s(\mathbf{u})$ are given in [29].

The Jacobi matrix $\mathbf{A}(\mathbf{u}, \mathbf{n}) = \sum_{i=1}^{d} n_i \mathbb{A}_i(\mathbf{u}) \in \mathbb{R}^{(d+2)\times(d+2)}$ can be diagonalized with

$$\mathbf{A}(\mathbf{u}, \mathbf{n}) = T\Lambda T^{-1},$$

where $\Lambda = \text{diag}(\lambda_i)$ is the diagonal matrix of eigenvalues of $\mathbf{A}(\mathbf{u}, \mathbf{n})$ and $T = (\mathbf{w}_1, \ldots, \mathbf{w}_7)$ is the matrix of eigenvectors $\mathbf{w}_i$ corresponding to the eigenvalues λ_i [11]. These eigenvalues of the Jacobi matrix are

$$\lambda_1 = \ldots = \lambda_d = \mathbf{v} \cdot \mathbf{n}, \quad \lambda_{d+1} = \mathbf{v} \cdot \mathbf{n} + a, \quad , \lambda_{d+2} = \mathbf{v} \cdot \mathbf{n} - a \tag{2.10}$$

where $a = \sqrt{\gamma p/\rho}$ denotes the speed of sound. This eigen-decomposition corresponds to the transformation to a set of characteristic variables, with $\mathbf{c} = T^{-1}\mathbf{u}$. Hence, the diagonalization of the Jacobi matrix can be interpreted as a decomposition into different waves. The eigenvectors in T represent the waves, the characteristic variables are the wave amplitudes, and the eigenvalues are the associated wave speeds [11].

The boundary Γ of Ω can be subdivided into an inflow part Γ_{in}, an outflow part Γ_{out} and a wall part Γ_{wall}, where Γ is a disjoint union of Γ_{in}, Γ_{out} and Γ_{wall}. The in- and outflow boundaries are defined according to the algebraic sign of $\mathbf{v} \cdot \mathbf{n}$, where $\mathbf{n}$ is the outer normal vector at Γ, in particular inflow at $\mathbf{v} \cdot \mathbf{n} < 0$ and outflow at $\mathbf{v} \cdot \mathbf{n} \geq 0$. Considering the signs of λ_i, for $i = 1, \ldots, d+2$, four cases of boundary conditions are distinguished:

- supersonic inflow at Γ_{in}: $\lambda_i < 0$, $i = 1, \ldots, d+2$,
- subsonic inflow at Γ_{in}: $\lambda_i < 0$, $i = 1, \ldots, d+1$, $\lambda_{d+2} > 0$,
- subsonic outflow at Γ_{out}: $\lambda_1 < 0$, $\lambda_i > 0$, $i = 2, \ldots, d+2$, and
- supersonic outflow at Γ_{out}: $\lambda_i > 0$, $i = 1, \ldots, d+2$.

Each eigenvalue smaller than zero corresponds to an inflow characteristic of the partial differential equation (PDE). The number of variables to be prescribed on the boundary depends on the number of inflow characteristics. Hence, a different number of boundary values and conditions are required for all of these cases. The vector $\mathbf{u}_\infty$ is described based on freestream conditions at the far field with

$$\mathbf{u}_\infty = \begin{pmatrix} \rho_\infty \\ \rho_\infty v_{1,\infty} \\ \vdots \\ \rho_\infty v_{d,\infty} \\ \rho_\infty E_\infty \end{pmatrix}. \tag{2.11}$$

The freestream conditions described the air before the aerodynamic body, where the body has no influence to deflect, slow down or compress the air. Moreover, in this approach the air is moving around the body incontrast to moving the body to air, hence the coordinate system is based around the aerodynamic body. The far field is the boundary of Ω which is chosen far away from the wall boundary Γ_{wall} inorder to be able to apply far field values and conditions, with the help of the freestream conditions, correctly. The boundary value function $\mathbf{u}_\Gamma(\mathbf{u})$ is defined case dependent at the far field boundary as follows

- supersonic inflow: $\mathbf{u}_\Gamma(\mathbf{u}) = \mathbf{u}_\infty$,
- subsonic inflow: $\mathbf{u}_\Gamma(\mathbf{u}) = \left(\rho_\infty, \rho_\infty v_{1,\infty}, \ldots, \rho_\infty v_{d,\infty}, \frac{p(\mathbf{u})}{\gamma-1} + \frac{\rho_\infty}{2}\|\mathbf{v}_\infty\|_2^2\right)^\top$,
- supersonic outflow: $\mathbf{u}_\Gamma(\mathbf{u}) = \mathbf{u}$,
- subsonic outflow: $\mathbf{u}_\Gamma(\mathbf{u}) = \left(\rho, \rho v_1, \ldots, \rho v_d, \frac{p_\infty}{\gamma-1} + \frac{\rho}{2}\|\mathbf{v}\|_2^2\right)^\top$.

Note that for the subsonic inflow definition of $\mathbf{u}_\Gamma(\mathbf{u})$ the pressure is evaluated from the flow field with equation (2.5), whereas all other values are taken from the freestream boundary condition $\mathbf{u}_\infty$ and vice versa for the subsonic outflow definition. For the subsonic outflow it is vice versa as described above. Finally the characteristic far field boundary condition imposes Dirichlet boundary conditions based on freestream conditions on characteristic inflow variables. No boundary conditions are imposed on characteristic outflow

variables. As an example Dirichlet conditions for $\mathbf{v}$ are of the form $\mathbf{v}(x,t) = \mathbf{v}_G(x,t)$ for $x \in \Gamma$ with a given function $\mathbf{v}_G$.

At the wall boundary Γ_{wall}, no-slip Dirichlet boundary conditions are prescribed on the velocity $\mathbf{v}$, hence, the velocity components of the boundary value function are set to zero, for the other components von Neuman boundary conditions are used [28],

$$\mathbf{u}_\Gamma(\mathbf{u}) = \begin{pmatrix} \rho \\ 0 \\ \vdots \\ 0 \\ \rho E \end{pmatrix}. \tag{2.12}$$

The viscous fluxes are defined as $\mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) = (\mathbf{f}_1^v, \ldots, \mathbf{f}_d^v) \in \mathbb{R}^{(d+2)\times d}$, with

$$\mathbf{f}_s^v(\mathbf{u}, \nabla\mathbf{u}) = (0, \tau_{s1}, \ldots, \tau_{sd}, \sum_{i=1}^{d} \tau_{si} v_i + \mathcal{K}\frac{\partial T}{\partial x_s})^\top, \quad s \in \underline{d}, \tag{2.13}$$

where $\mathcal{K}$ is defined in (2.6) and the viscous stress tensor is given by $\tau_{si} = -\frac{2}{3}\mu\nabla\cdot\mathbf{v}\delta_{si} + 2\mu\mathbb{D}(\mathbf{v})_{si}$, $s, i \in \underline{d}$ while applying (2.3). The viscous stress tensor describes the friction between the fluid and the surface of an element. At adiabatic wall boundaries the last component in $\mathbf{f}_s^v$ reduces to $\sum_{i=1}^{d} \tau_{si} v_i$. The viscous fluxes are linear with respect to the gradient $\nabla\mathbf{u}$ and can be rewritten as follows

$$\mathbf{f}_s^v(\mathbf{u}, \nabla\mathbf{u}) = \sum_{l=1}^{d} \mathbb{G}_{sl}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x_l},\ s \in \underline{d}. \tag{2.14}$$

In the two and three dimensions, the matrices $\mathbb{G}_{sl}$ are given in [29]
With that in mind (2.7) can be written as:

$$\sum_{s=1}^{d}\left(\mathbb{A}_s(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x_s} - \sum_{l=1}^{d}\frac{\partial}{\partial x_s}(\mathbb{G}_{sl}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x_l})\right) = 0 \text{ in } \Omega, \tag{2.15}$$

with (2.9) and (2.14). Please note that this flux formulation is a common starting point for the discretization the Navier-Stokes equations or similar equations and has been used in many publications in the field of CFD [10, 26, 27, 56, 76, 78, 79].

It remains to define the viscosity coefficient μ. To this end, the Reynolds number (Re) [11] is introduced as follows

$$\text{Re} = \frac{\rho_\infty \|\mathbf{v}_\infty\|_2 L}{\mu_\infty}, \tag{2.16}$$

where $\rho_\infty, \mathbf{v}_\infty$ and μ_∞ are values at the far field $\Gamma_{\text{in}} \cup \Gamma_{\text{out}}$. Here, L is a reference length of the body submerged in the fluid, e.g. the chord of an airfoil.

The equations (2.7) are non-dimensionalized based on the reference pressure, density and temperature given by the corresponding far field values. Thereby, the non-dimensionalized pressure and density at the far field are given by $p_\infty = \rho_\infty = 1$. Furthermore μ_∞ is given by

$$\mu_\infty = \frac{\sqrt{\gamma}\mathsf{M}_\infty L}{\mathsf{Re}}, \tag{2.17}$$

where M_∞ denotes the Mach number at the far field. The Mach number is defined as $\mathsf{M} = \frac{\|\mathbf{v}\|_2}{a}$, where $a = \sqrt{\gamma p/\rho}$ is the speed of sound for air. Moreover, a Mach number of one means that the velocity of the flow equals the speed of sound, which is $340.29\,\mathrm{m/s}$ for air at sea level. In this work, the Mach number prescribed at the far field is always smaller than one, since subsonic flows are the main focus of this work.

Finally the viscosity coefficient μ_∞ is modified depending on the local temperature according to Sutherland's law [67] which describes the local changes from constant μ_∞ to μ triggered by heat fluctuations in the flow.

As can be seen from (2.17) the Reynolds number is large in comparison to the Mach number if the viscosity of the flow field is very small. Hence for a small viscosity coefficient μ the viscous stress tensor τ_{si} and thus the viscous flux is small. Due to this most of the flow field is convection dominated for a higher Reynolds number flow. The velocity of the flow field is then close to the velocity prescribed at the far field except in a very thin region close to the boundary Γ_{wall}. The Reynolds number can therefore be seen as an indicator for viscosity in a desired flow field. Note that in Section 5.1 only the Mach number, the Reynolds number and the angle of attack at the far field will be provided for each test case.

2.3 Reynolds-averaged Navier-Stokes equations

In this section, the RANS equations will be introduced. In contrast to a laminar flow the molecules in a turbulent flow move in a chaotic fashion. Hence, the turbulence is a chaotic motion which causes a mixing of the various layers of a fluid. Due to this chaotic motion the simulation of turbulent flows is still a challenge [11]. Despite the performance of modern computers and cluster systems, a direct numerical simulation of turbulence by the time-dependent Navier-Stokes equations (2.1) known as the direct numerical simulation (DNS) [16] is applicable only to relatively simple flow problems at low Reynolds numbers [11]. To overcome this problem mass-weighted time-averaging is applied on the complete equation system (2.1) resulting in the RANS equations. Because of the averaging procedure the equations become stationary and new terms occur, which

are the turbulent heat flux and the so-called Reynolds stress tensor [16], the trace of which is connected to the turbulence kinetic energy. Due to the new terms in the RANS equations it is again necessary to add additional relations to close the equations.

The Boussinesq approximation results in the use of an eddy viscosity to model the turbulence Reynolds stresses [11]. This gives a proportionality factor between the Reynolds stresses and the velocity gradient, namely the turbulent eddy viscosity μ_t. Moreover, the turbulent Prandtl number Pr_t is defined as the ratio between the turbulent momentum eddy diffusivity and the turbulent heat transfer eddy diffusivity. This is in analogy to the non-turbulent Prandtl number Pr introduced in Section 2.1.

With these two approximations only two new variables, the turbulent kinetic energy k and the turbulent eddy viscosity μ_t, remain to be modeled in the RANS equations. Therefore it is required to introduce a so-called RANS turbulence model, which are still the backbone of the numerical simulation of technical flow problems. The main goal of this work is to solve the RANS equations in combination with a turbulence model efficiently.

In the following two different turbulence models are introduced which both can be written in conservative form based on convective and viscous fluxes $\mathcal{F}^c$ and $\mathcal{F}^v$, and additional source terms $\mathcal{S}(\mathbf{u}, \nabla\mathbf{u})$ as follows,

$$\nabla \cdot (\mathcal{F}^c(\mathbf{u}) - \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u})) - \mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) = 0. \tag{2.18}$$

The number of equations in (2.18) varies for different space dimensions and turbulence models and will be addressed in the upcoming subsections.

2.3.1 The steady-state RANS-$k\omega$ equations

The steady-state RANS and Wilcox $k\omega$ (RANS-$k\omega$) turbulence model equations [78, 79] can be written as in (2.18) with the vector of conservative variables

$$\mathbf{u} = (\rho, \rho v_1, \ldots, \rho v_d, \rho E, \rho k, \rho\omega)^\top,$$

where ρ denotes the density, $\mathbf{v} = (v_1, \ldots, v_d)^\top$ the velocity vector, E the total energy, k the turbulent kinetic energy and ω the turbulence dissipation rate. This model sets the eddy viscosity μ_t from the Boussinesq approximation to

$$\mu_t := \frac{\rho k}{\omega}, \tag{2.19}$$

and includes two transport equations for the turbulence kinetic energy k and the specific dissipation rate ω. Furthermore, as mentioned before a turbulent Prandtl number Pr_t is in-

troduced [55], for dry air $\mathrm{Pr}_t = 0.9$. For the RANS-$k\omega$ equations the pressure is determined by the following equation of state

$$p = (\gamma - 1)\left(\rho E - \frac{\rho}{2}\|\mathbf{v}\|_2^2 - \rho k\right). \tag{2.20}$$

This definition is the same as in (2.5), modified to include an additional k-term. Then the convective flux is given by $\mathcal{F}^c(\mathbf{u}) = (\mathbf{f}_1^c, \ldots, \mathbf{f}_d^c) \in \mathbb{R}^{(d+4)\times d}$ with

$$\mathbf{f}_s^c(\mathbf{u}) = \mathbf{u}v_s + (0, \delta_{s1}, \ldots, \delta_{sd}, v_s, 0, 0)^\top p_{\text{eff}}, \tag{2.21}$$

where p_{eff} denotes the effective pressure given by $p_{\text{eff}} = p + \frac{2}{3}\rho k$. This is the same formulation as in (2.8) for the steady-state Navier-Stokes equations, only the pressure p is changed to p_{eff} and two additional rows for the turbulence model equations are introduced.

The viscous flux is given by $\mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) = (\mathbf{f}_1^v, \ldots, \mathbf{f}_d^v) \in \mathbb{R}^{(d+4)\times d}$ with

$$\mathbf{f}_s^v(\mathbf{u}, \nabla\mathbf{u}) = \begin{pmatrix} 0 \\ \tilde{\tau}_{s1} \\ \vdots \\ \tilde{\tau}_{sd} \\ \sum_{i\in\underline{d}} \tilde{\tau}_{si}v_i + c_p\left(\frac{\mu}{\mathrm{Pr}} + \frac{\mu_t}{\mathrm{Pr}_t}\right)\frac{\partial T}{\partial x_s} + (\mu + \sigma_k\mu_t)\frac{\partial k}{\partial x_s} \\ (\mu + \sigma_k\mu_t)\frac{\partial k}{\partial x_s} \\ (\mu + \sigma_\omega\mu_t)\frac{\partial \omega}{\partial x_s} \end{pmatrix} \tag{2.22}$$

and the turbulent viscous stress tensor $\tilde{\tau}_{si} = (\mu + \mu_t)(-\frac{2}{3}\nabla\cdot\mathbf{v}\delta_{si} + 2\mathbb{D}(\mathbf{v})_{si})$, $s, i \in \underline{d}$. Hence, $\tilde{\tau}_{si} = \tau_{si} + \frac{\mu_t}{\mu}\tau_{si}$ with τ_{si} as defined in Section 2.2 for the steady-state Navier-Stokes equations. Here σ_k and σ_ω are parameters of the turbulence model given by $\sigma_k = \sigma_\omega = \frac{1}{2}$ [78, 79]. Note, that the viscous flux of the RANS-$k\omega$ equations differs from the viscous flux in (2.13) only through the addition of the turbulent Prandtl number Pr_t, the turbulent eddy viscosity μ_t and two additional transport equations for k and ω.

The additional source terms are given by

$$\mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) = (\underbrace{0, \ldots, 0}_{d+2}, \sum_{i,j\in\underline{d}} \tau_{ij}^R\frac{\partial v_i}{\partial x_j} - \beta_k\rho k\omega, \sum_{i,j\in\underline{d}} \alpha_\omega\frac{\omega}{k}\tau_{ij}^R\frac{\partial v_i}{\partial x_j} - \beta_\omega\rho\omega^2)^\top,$$

with $\tau_{ij}^R = 2\mu_t\mathbb{D}(\mathbf{v})_{ij} - \frac{2}{3}(\mu_t\nabla\cdot\mathbf{v} + \rho k)\delta_{ij}$ and $\beta_k, \alpha_\omega, \beta_\omega$ are parameters of turbulence model, given by $\beta_k = \frac{9}{100}, \alpha_\omega = \frac{5}{9}$ and $\beta_\omega = \frac{3}{40}$ [78, 79]. Moreover, $\tilde{\tau}_{si}$ can be written as $\tilde{\tau}_{si} = \tau_{si} + \tau_{si}^R + \frac{2}{3}\rho k\delta_{si}$, $s, i \in \underline{d}$. Note that, the source terms only take effect in the additional transport equations of the $k\omega$-turbulence model.

Similar to [5, 18] the equations are now considered in terms of the auxiliary variable $\tilde{\omega} = \ln(\omega)$ instead of ω for a more moderate near-wall behavior of the variable. The

use of the logarithm of turbulence variables has been introduced in [5] and results in an equivalent reformulation of the Wilcox $k\omega$ equations [26]. Near the wall ω is close to infinity and the transformation to $\tilde{\omega}$ weakens this effect. Additionally, this variable transformation guarantees the positivity of ω which helps in the numerical discretization of the RANS equations in Chapter 3.

Now every ω will be replaced by $e^{\tilde{\omega}}$. Because of that a new vector of conservative variables $\mathbf{u} = (\rho, \rho v_1, \ldots, \rho v_d, \rho E, \rho k, \rho\tilde{\omega})^\top$ is given. Applying the ∇ operator and the transformation on ω one gets

$$\nabla\omega = \nabla e^{\tilde{\omega}} = e^{\tilde{\omega}}\nabla\tilde{\omega} = \omega\nabla\tilde{\omega},$$

and thus for the diffusive and convective part of the ω-transport equation of (2.18)

$$\nabla\cdot((\mu+\sigma_\omega\mu_t)\nabla\omega) = e^{\tilde{\omega}}\left((\mu+\sigma_\omega\mu_t)\nabla\tilde{\omega}\cdot\nabla\tilde{\omega} + \nabla\cdot(\mu+\sigma_\omega\mu_t)\nabla\tilde{\omega}\right)$$

and

$$\nabla\cdot(\rho\omega\mathbf{v}) = e^{\tilde{\omega}}\left(\nabla\cdot(\rho\mathbf{v}) + \rho\mathbf{v}\cdot\nabla\tilde{\omega}\right),$$

respectively. Taking the equivalent steady-state equation to (2.1) into account, it follows

$$\nabla\cdot(\rho\omega\mathbf{v}) = e^{\tilde{\omega}}\nabla\cdot(\rho\tilde{\omega}\mathbf{v}).$$

In addition to that the ω-transport equation will be divided by $\omega = e^{\tilde{\omega}} \neq 0$, with that it is possible to state the RANS-$k\omega$ equations in flux formulation in combination with the $\ln(\omega)$-transformation. The transport equation for the specific dissipation rate ω can now be rewritten as

$$\nabla\cdot(\rho\tilde{\omega}\mathbf{v} - (\mu+\sigma_\omega\mu_t)\nabla\tilde{\omega}) = \frac{\alpha_\omega}{k}\sum_{i,j\in\underline{d}}\tau_{ij}^R\frac{\partial v_i}{\partial x_j} - \beta_\omega\rho e^{\tilde{\omega}} + (\mu+\sigma_\omega\mu_t)\nabla\tilde{\omega}\cdot\nabla\tilde{\omega}. \tag{2.23}$$

Furthermore, a limitation of $\tilde{\omega}$ is introduced. Let $\tilde{\omega}_l = \max[\tilde{\omega}, \tilde{\omega}_{l_0}]$ where $\tilde{\omega}_{l_0}$ fulfills following realizability conditions for the turbulent stresses,

$$e^{\tilde{\omega}_{l_0}} - \frac{3}{2}\frac{\tau_{ii}}{\mu} \geqslant 0,\ \forall\, i\in\underline{d}, \tag{2.24}$$

$$e^{2\tilde{\omega}_{l_0}} - \frac{3}{2}\frac{\tau_{ii}+\tau_{jj}}{\mu}e^{\tilde{\omega}_{l_0}} + \frac{9}{4}\frac{\tau_{ii}\tau_{jj}-\tau_{ij}^2}{\mu^2} \geqslant 0,\ \forall\, i,j\in\underline{d},\ i\neq j, \tag{2.25}$$

equivalent to the realizability conditions given in [5, 18]. In analogy to [5] some $\tilde{\omega}$ are replaced by $\tilde{\omega}_l$ in (2.18). In particular, $\mu_t = \frac{\rho k}{\omega}$ is replaced by $\mu_t := \frac{\rho k}{\tilde{\omega}_l}$. Note, that the limitations on $\tilde{\omega}$ avoid unphysical values and has been found in [5, 18, 26] to have an stabilizing effect on the numerical scheme.

This concludes the introduction of the steady-state RANS-$k\omega$ equations in flux formulation. The system of equations is given by (2.18), with the vector of conservative variables $\mathbf{u} = (\rho, \rho v_1, \ldots, \rho v_d, \rho E, \rho k, \rho\tilde{\omega})^\top$. Applying the transformation of ω and the limitation of $\tilde{\omega}$ the convective flux is given by $\mathcal{F}^c(\mathbf{u}) = (\mathbf{f}_1^c, \ldots, \mathbf{f}_d^c) \in \mathbb{R}^{(d+4)\times d}$ with

$$\mathbf{f}_s^c(\mathbf{u}) = \mathbf{u}v_s + (0, \delta_{s1}, \ldots, \delta_{sd}, v_s, 0, 0)^\top p_{\text{eff}}. \tag{2.26}$$

The viscous flux is given by $\mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) = (\mathbf{f}_1^v, \ldots, \mathbf{f}_d^v)$ with

$$\mathbf{f}_s^v(\mathbf{u}, \nabla\mathbf{u}) = \begin{pmatrix} 0 \\ \tilde{\tau}_{s1} \\ \vdots \\ \tilde{\tau}_{sd} \\ \sum_{i\in\underline{d}} \tilde{\tau}_{si} v_i + c_p \left(\frac{\mu}{\text{Pr}} + \frac{\mu_t}{\text{Pr}_t}\right) \frac{\partial T}{\partial x_s} + (\mu + \sigma_k\mu_t)\frac{\partial k}{\partial x_s} \\ (\mu + \sigma_k\mu_t)\frac{\partial k}{\partial x_s} \\ (\mu + \sigma_\omega\mu_t)\frac{\partial\tilde{\omega}}{\partial x_s} \end{pmatrix}.$$

The additional source terms are given by

$$\mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \sum_{i,j\in\underline{d}} \tau_{ij}^R \frac{\partial v_i}{\partial x_j} - \beta_k \rho k e^{\tilde{\omega}_l} \\ \sum_{i,j\in\underline{d}} \frac{\alpha_\omega}{k} \tau_{ij}^R \frac{\partial v_i}{\partial x_j} + (\mu + \sigma_\omega\mu_t) \sum_{s\in\underline{d}} \frac{\partial\tilde{\omega}}{\partial x_s}\frac{\partial\tilde{\omega}}{\partial x_s} - \beta_\omega \rho e^{\tilde{\omega}_l} \end{pmatrix}.$$

Due to the k-variable occurring in the energy equation and missing boundary values for the turbulence variables, some new boundary values need to be defined. In particular, at the far field boundary Dirichlet conditions for k and $\tilde{\omega}$ are given by

$$k_\infty = \frac{d}{2}(\text{Tu}\,\mathbf{v}_\infty)^2, \tag{2.27}$$

$$\tilde{\omega}_\infty = \ln\left(\frac{\rho_\infty k_\infty}{r\mu_\infty}\right), \tag{2.28}$$

where Tu is the turbulence intensity, $r = \frac{\mu_{t,\infty}}{\mu_\infty}$ is the ratio between the turbulent and molecular viscosity and d the space dimension. In this work the following values are used, $\text{Tu} = 0.001$ and $r = 0.01$. The definition of $\mathbf{u}_\Gamma(\mathbf{u})$ is the same as seen in Section 2.2, with

$$\mathbf{u}_\infty = (\rho_\infty, \rho_\infty v_{1,\infty}, \ldots, \rho_\infty v_{d,\infty}, \rho_\infty E_\infty, \rho_\infty k_\infty, \rho_\infty\tilde{\omega}_\infty)^\top. \tag{2.29}$$

At the wall boundary Γ_{wall}, boundary conditions similar to (2.12) are prescribed with

$$\mathbf{u}_\Gamma(\mathbf{u}) = (\rho, 0, \ldots, 0, \rho E, 0, \rho\tilde{\omega}_{\text{wall}})^\top. \tag{2.30}$$

Near the wall, the turbulent kinetic energy k vanishes, therefore, the boundary values of k on Γ_{wall} is set to zero. The analytical solution of $\tilde{\omega}$ on Γ_{wall} is infinity, $\tilde{\omega}_{\text{wall}} = \infty$. Hence, for the discontinuous Galerkin (DG) discretization of these equations $\tilde{\omega}_{\text{wall}}$ at the wall needs to be approximated, more details on this issue will be given in Section 3.3.

2.3.2 The steady-state RANS-SA equations

In this section the RANS equations in combination with another turbulence model are introduced. As in the Section 2.3.1 the main target is how to model the eddy viscosity μ_t and how this modeling choice effects the other equations. A modification of the original Spalart–Allmaras turbulence model [66], namely the negative Spalart–Allmaras turbulence model (SA), will be presented in this work and was introduced in [57]. The SA turbulence model is a one-equation model which means that only one additional equation to the RANS equations is necessary. In comparison the Wilcox $k\omega$ turbulence model is a two equation model, as seen in Section 2.3.1. The steady-state RANS-SA equations [57, 66] can also be written in conservative form (2.18) with the vector of conservative variables

$$\mathbf{u} = (\rho, \rho v_1, \ldots, \rho v_d, \rho E, \rho\tilde{\nu})^\top,$$

where $\tilde{\nu}$ is a model variable of the SA turbulence model. This model sets the eddy viscosity μ_t to

$$\mu_t = \begin{cases} \rho\tilde{\nu} f_{v_1}, & \text{if } 0 < \tilde{\nu}, \\ 0, & \text{else}, \end{cases} \tag{2.31}$$

with $f_{v_1} = \frac{\mathcal{X}^3}{\mathcal{X}^3 + c_{v_1}^3}$, $\mathcal{X} = \frac{\tilde{\nu}}{\nu}$, where $\nu = \frac{\mu}{\rho}$ is the kinematic (molecular, i.e. non-turbulent) viscosity, and a constant $c_{v_1} = 7.1$. For the RANS-SA equations the same definition for pressure is used as for the compressible Navier-Stokes equations or the RANS-$k\omega$ equations with $k = 0$:

$$p = (\gamma - 1)\left(\rho E - \frac{\rho}{2}\|\mathbf{v}\|_2^2\right). \tag{2.32}$$

A main difference between the negative Spalart–Allmaras turbulence model and the Wilcox $k\omega$ turbulence model, as seen in subsection 2.3.1, is that there is no direct coupling between the additional transport equation for $\tilde{\nu}$ and the energy equation. This simplifys $\mathcal{F}^c(\mathbf{u})$ and $\mathcal{F}^v(\mathbf{u})$ in comparison to the RANS-$k\omega$ case. The turbulent kinetic energy k which results from the Boussinesq approximation is set to zero in the SA turbulence

model. This changes the energy equation of equations (2.21) & (2.22) of Subsection 2.3.1 and every term which includes k in comparison to the $k\omega$ turbulence model. With this in mind the only coupling from the additional SA transport equation to the main equations is due to μ_t. Therefore, in this section the main focus will be on the additional equation for $\tilde{\nu}$. Also the source terms are only appearing in this additional transport equation. This is similar to the RANS-$k\omega$ equations where additional source terms only appeared in the $k\omega$ transport equations.

The negative Spalart–Allmaras turbulence model has different contributions to the last component of $\mathcal{F}^c(\mathbf{u})$, $\mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u})$ and $\mathcal{S}(\mathbf{u}, \nabla\mathbf{u})$. Some of these contributions depend on the algebraic sign of $\tilde{\nu}$. Hence the convective flux is given by $\mathcal{F}^c(\mathbf{u}) = (\mathbf{f}_1^c, \ldots, \mathbf{f}_d^c) \in \mathbb{R}^{(d+3)\times d}$ with

$$\mathbf{f}_s^c(\mathbf{u}) = \mathbf{u}v_s + (0, \delta_{s1}, \ldots, \delta_{sd}, v_s, 0)^\top p. \tag{2.33}$$

The viscous flux is given by $\mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) = (\mathbf{f}_1^v, \ldots, \mathbf{f}_d^v) \in \mathbb{R}^{(d+3)\times d}$ with

$$\mathbf{f}_s^v(\mathbf{u}, \nabla\mathbf{u}) = \begin{pmatrix} 0 \\ \tilde{\tau}_{s1} \\ \vdots \\ \tilde{\tau}_{sd} \\ \sum_{i\in\underline{d}} \tilde{\tau}_{si}v_i + c_p\left(\frac{\mu}{\mathsf{Pr}} + \frac{\mu_t}{\mathsf{Pr}_t}\right)\frac{\partial T}{\partial x_s} \\ \frac{\rho}{\sigma}(\nu + f_n\tilde{\nu})\frac{\partial\tilde{\nu}}{\partial x_s} \end{pmatrix}, \tag{2.34}$$

and the turbulent viscous stress tensor $\tilde{\tau}_{si} = (\mu + \mu_t)(-\frac{2}{3}\nabla\cdot\mathbf{v}\delta_{si} + 2\mathbb{D}(\mathbf{v})_{si})$, $s, i \in \underline{d}$. Also $\sigma = \frac{2}{3}$ as a model constant and the function f_n is given by

$$f_n = \begin{cases} 1, & \text{if } \tilde{\nu} > 0, \\ \frac{c_n + \mathcal{X}^3}{c_n - \mathcal{X}^3}, & \text{else}, \end{cases}$$

with $c_n = 16$.

The source term are defined as follows

$$\mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathsf{P} - \mathsf{D} + \frac{\rho c_{b2}}{\sigma}\sum_{s\in\underline{d}}\frac{\partial\tilde{\nu}}{\partial x_s}\frac{\partial\tilde{\nu}}{\partial x_s} - \frac{1}{\sigma}(\nu + f_n\tilde{\nu})\sum_{s\in\underline{d}}\frac{\partial\rho}{\partial x_s}\frac{\partial\tilde{\nu}}{\partial x_s} \end{pmatrix}, \tag{2.35}$$

where

$$\mathsf{P} = \begin{cases} \rho c_{b1}(1 - f_{t2})\hat{S}\tilde{\nu} & \text{if } \tilde{\nu} \geq 0, \\ \rho c_{b1}(1 - c_{t3})\tilde{S}\tilde{\nu} & \text{else}, \end{cases}$$

$$\mathsf{D} = \begin{cases} \rho\left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right)\left(\frac{\tilde{\nu}}{\mathsf{d}}\right)^2 & \text{if } \tilde{\nu} \geq 0, \\ -\rho c_{w1}\left(\frac{\tilde{\nu}}{\mathsf{d}}\right)^2 & \text{else}, \end{cases}$$

with d as distance from a field point to the nearest wall.

Now follows a list of unknown constant and function involved:

$$c_{b1} = 0.1355, \quad f_{t2} = c_{t3}\exp\left(-c_{t4}\mathcal{X}^2\right), \quad c_{t3} = 1.2, \quad c_{t4} = 0.5,$$
$$c_{w2} = 0.3, \quad f_{v2} = 1 - \frac{\mathcal{X}}{1+\mathcal{X} f_{v1}}, \quad \tilde{S} = \sqrt{2W_{ij}W_{ij}}, \quad W_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right),$$
$$c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma}, \quad c_{b2} = 0.622, \quad f_w = g_1\left[\frac{1+c_{w3}^6}{g_1^6+c_{w3}^6}\right]^{1/6}, \quad c_{w3} = 2,$$
$$\kappa = 0.41, \quad g_1 = r_1 + c_{w2}(r_1^6 - r_1) \quad r_1 = \min\left[\frac{\tilde{\nu}}{\hat{S}\kappa^2\mathsf{d}^2}, 10\right].$$

Note that the above list only states functions and constants which are used solely in the source terms of the RANS-SA model.

To avoid possible numerical problems, the term $\hat{S}$ in P must never be allowed to reach zero or go negative. A common solution for this problem is applied in the following and was first introduced in [57]. For simplicity let

$$\bar{S} := \frac{\tilde{\nu}}{\kappa^2\mathsf{d}^2} f_{v_2}.$$

Then $\hat{S}$ is defined as follows

$$\hat{S} = \begin{cases} \tilde{S} + \bar{S}, & \text{if } \bar{S} \geq -c_{v2}\tilde{S} \\ \tilde{S} + \frac{\tilde{S}(c_{v2}^2\tilde{S} + c_{v3}\bar{S})}{(c_{v3}-2c_{v2})\tilde{S} - \bar{S}}, & \text{else}, \end{cases}$$

where $c_{v2} = 0.7$ and $c_{v3} = 0.9$, also note $\tilde{S} \geq 0$.

The definition of $\mathbf{u}_\Gamma(\mathbf{u})$ is the same in Section 2.2, with an additional boundary condition for $\rho\tilde{\nu}$. In particular, at the far field $\rho\tilde{\nu}$ is given by

$$\rho_\infty\tilde{\nu}_\infty = \rho_\infty\mathcal{X}_\infty\nu_\infty, \tag{2.36}$$

where $\nu_\infty = \frac{\mu_\infty}{\rho_\infty}$, $\mathcal{X}_\infty = 4$ and μ_∞ is computed with the help of Sutherland's law [67]. At the Γ_{wall} a no-slip boundary conditions is defined as

$$\mathbf{u}_\Gamma(\mathbf{u}) = (\rho, 0, \ldots, 0, \rho E, 0)^\top. \tag{2.37}$$

3 Discontinuous Galerkin discretization

Finite element (FE) methods are usually defined over a piecewise polynomial space, which includes polynomials up to a certain degree $p \in \mathbb{N}$. The choice of a discrete space of ansatz and test functions is important as it might enable the discretization on meshes consisting of general polyhedral elements. A general polyhedral element mesh is obtained as the result of the agglomeration of a standard unstructured mixed-element mesh covering the whole computational domain Ω. The properties of such a computational mesh for higher order finite element methods will be handled in Section 3.5. The possibility to formulate the discontinuous Galerkin (DG) discretization of the governing equations, e.g. equation (2.18), on general polyhedra enables the incorporation of agglomerated meshes as coarse levels into a geometric multigrid algorithm.

3.1 Triangulation

For a better understanding of the solution spaces considered in of finite element methods, a very short introduction is given. Let the domain Ω be open, bounded, and connected. Let the boundary $\Gamma := \partial\Omega$ consist of different parts which are graphs of smooth functions and at the interface between different parts interior angles are bounded away from zero. Furthermore, let Ω be a Lipschitz domain [17]. The Hilbert space $L^2(\Omega)$ denotes the space of all measurable functions from Ω to $\mathbb{R}$ with

$$\|f\|_{L^2} := \left(\int_\Omega f^2 \, \mathrm{d}V \right)^{\frac{1}{2}} < \infty. \tag{3.1}$$

Note that the scalar product $\langle \cdot, \cdot \rangle_{L^2}$ in L^2 is defined as $\langle f, g \rangle_{L^2} := \int_\Omega fg \, \mathrm{d}V$ for $f, g \in L^2(\Omega)$. Let $C_0^\infty(\Omega)$ be the space of all functions which are arbitrarily often differentiable in Ω and equal to zero on Γ. The weak derivative is defined, with the help of Green's formula [17], as follows.

Definition 3.1:

Let $u \in L^2(\Omega)$ and $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{N}_0^n$ be a multi-index of order $|\alpha| > 0$ with $|\alpha| = \alpha_1 + ... + \alpha_n$. If there exists $v \in L^2(\Omega)$ such that

$$\int_\Omega v(\mathbf{x})\sigma(\mathbf{x})\,\mathrm{d}\mathbf{x} = (-1)^{|\alpha|}\int_\Omega u(\mathbf{x})D^\alpha\sigma(\mathbf{x})\,\mathrm{d}\mathbf{x} \;\forall\; \sigma \in C_0^\infty(\Omega), \tag{3.2}$$

then v is called the α-th weak derivative of u and is denoted by $D^\alpha u = \frac{\partial^{|\alpha|}u}{\partial x_1^{\alpha_1}...\partial x_n^{\alpha_n}} =: v$ with $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{x} = (x_1, ..., x_n)$.

Now a Sobolev space is introduced [17, 48].

Definition 3.2:

The Sobolev space $H^m(\Omega)\,, m \in \mathbb{N}$ consists of all functions in $L^2(\Omega)$ for which all α-th weak derivatives with $|\alpha| \leq m$ exist:

$$H^m(\Omega) := \{u \in L^2(\Omega) \,:\, D^\alpha u \in L^2(\Omega),\; \forall\; 0 < |\alpha| \leq m\}.$$

Note that all Sobolev spaces are Banach spaces [43]. Therefore, all Sobolev spaces are equipped with a norm and are complete with respect to that norm.

Assume that the domain Ω is subdivided into $n_t \in \mathbb{N}$ open and disjoint subdomains κ. The set of all κ is representing the computational mesh $\mathcal{T}_h$ with

$$\overline{\Omega} = \bigcup_{\kappa\in\mathcal{T}_h} \overline{\kappa},\; \kappa_i \cap \kappa_j = \emptyset \;\forall\; i, j \in \underline{n_t},\; i \neq j.$$

Now, it is possible to define a broken Sobolev space. This space is needed because for discontinuous Galerkin discretizations it is necessary to allow functions in Ω to be discontinuous across element interfaces $\partial\kappa^+ \cap \partial\kappa^- \neq \emptyset$ with $\kappa^+, \kappa^- \in \mathcal{T}_h$.

Definition 3.3:

The broken Sobolev space $H^{m,b}(\mathcal{T}_h)$ denotes the space of L^2-functions on Ω whose restriction to each element κ belongs to the Sobolev space $H^m(\kappa)$, i.e.

$$H^{m,b}(\mathcal{T}_h) := \{\mathbf{v} \in [L^2(\Omega)]^n \,:\, \mathbf{v}|_\kappa \in [H^m(\kappa)]^n \,\forall\, \kappa \in \mathcal{T}_h\},$$

where $n = d + 4$ for the RANS-$k\omega$ equations and $n = d + 3$ for the RANS-SA equations. Here $d = 2$ or $d = 3$ denotes the space dimension of the problem.

A broken Sobolev space is the baseline for all DG discretizations. Considering polynomial functions on every element κ gives a discrete finite element space which is a subspace of a

broken Sobolev space $H^{m,b}(\mathcal{T}_h)$ on the given mesh; which polynomial functions to choose will be handled in Section 3.4.

A weak formulation of (2.18) in a broken Sobolev space can be stated as follows:

Find $\mathbf{u} \in H^{m,b}(\mathcal{T}_h)$ such that

$$N(\mathbf{u}, \mathbf{v}) = 0 \quad \forall \mathbf{v} \in H^{m,b}(\mathcal{T}_h), \tag{3.3}$$

with $N(\mathbf{u}, \mathbf{v}) = \sum_{\kappa \in \mathcal{T}_h} \int_\kappa (\nabla \cdot (\mathcal{F}^c(\mathbf{u}) - \mathcal{F}^v(\mathbf{u}, \nabla \mathbf{u})) - \mathcal{S}(\mathbf{u}, \nabla \mathbf{u})) \cdot \mathbf{v} \, \mathrm{d}V$.

A DG discretization of this problem is based on the following idea. Consider a finite dimensional subspace $\mathcal{V}_h^p(\mathcal{T}_h) \subset H^{m,b}(\mathcal{T}_h)$ and the finite dimensional variational problem:

$$\text{Find } \mathbf{u}_h \in \mathcal{V}_h^p(\mathcal{T}_h) \text{ such that } N_h(\mathbf{u}_h, \mathbf{v}_h) = 0 \quad \forall \mathbf{v}_h \in \mathcal{V}_h^p(\mathcal{T}_h). \tag{3.4}$$

This problem is called the DG discretization of $N(\mathbf{u}, \mathbf{v}) = 0$. The key point here is that $\mathcal{V}_h^p(\mathcal{T}_h)$ is considered of finite dimension $d_\mathcal{V} := \dim(\mathcal{V}_h^p(\mathcal{T}_h))$ and because of that there exists a finite basis $\{\mathbf{v}_i\}_{d_\mathcal{V}}$ of $\mathcal{V}_h^p(\mathcal{T}_h)$. Therefore, every $\mathbf{u}_h$ in $\mathcal{V}_h^p(\mathcal{T}_h)$ can be written as $\mathbf{u}_h := \sum_{i=0}^{d_\mathcal{V}} a_i \, \mathbf{v}_i$ with $\mathbf{v}_i \in \{\mathbf{v}_i\}_{d_\mathcal{V}}$, $a_i \in \mathbb{R}$. Due to the finite number of basis functions $\mathbf{v}_h$ in (3.4), the discrete problem (3.4) will result in a finite number of equations, which can be solved resulting in a discrete solution $\mathbf{u}_h \in \mathcal{V}_h^p(\mathcal{T}_h)$.

For a DG discretization space $\mathcal{V}_h^p(\mathcal{T}_h)$, polynomial functions of degree p on every $\kappa \in \mathcal{T}_h$ are used. Hence, the DG discretization space can be defined as

$$\mathcal{V}_h^p(\mathcal{T}_h) = \left\{ \mathbf{v}_h \in [L^2(\Omega)]^n \, : \, \mathbf{v}_h|_\kappa \in [\mathcal{P}_p]^n \right\}, \tag{3.5}$$

where $\mathcal{P}_p$ denotes the space of polynomial functions of degree at most p and $n = d + 4$ for the RANS-$k\omega$ equations or $n = d + 3$ for the RANS-SA equations, where d denotes the space dimension of the problem.

For model problems, e. g. Poisson equation, and some DG methods the error scales as $||\mathbf{u}_h - \mathbf{u}||_{L^2(\Omega)} \sim h^{p+1}$, if the solution is sufficiently smooth, where $\mathbf{u}_h \in \mathcal{V}_h^p(\mathcal{T}_h)$ is the numerical solution, $\mathbf{u}$ the analytical solution, and h the characteristic size of all $\kappa \in \mathcal{T}_h$. Here $p + 1$ is called the order of the DG method.

How to transform (3.3) to arrive at the discretization is explained next. The basic idea of using Gauss theorem [17] to get rid of the second order derivatives in the term $(\nabla \cdot (-\mathcal{F}^v(\mathbf{u}, \nabla \mathbf{u}))$ and transform the term $(\nabla \cdot \mathcal{F}^c(\mathbf{u}))$ in (3.3) is always the same for integral problems and will be applied here as well. Due to the possible discontinuity of the discrete solution $\mathbf{v}_h \in \mathcal{V}_h^p(\mathcal{T}_h)$ between two neighbor elements κ^+, κ^-, jump operators are defined as follows.

Definition 3.4:

Let Γ be the boundary of Ω and Γ_I the set of all interior inter-element faces in $\mathcal{T}_h$. Consider a vector-valued function $\mathbf{v} \in H^{m,b}(\mathcal{T}_h)$, a matrix-valued function $\underline{\mathbf{v}} \in [H^{m,b}(\mathcal{T}_h)]^d$ and $\mathbf{n}^i$ the outward unit normal vector of κ^i for $i = +, -$.

Then the mean values are defined by

$$\begin{aligned} \{\!\{\mathbf{v}\}\!\} &:= \tfrac{1}{2}(\mathbf{v}^+ + \mathbf{v}^-) \quad \text{on } \Gamma_I, \\ \{\!\{\underline{\mathbf{v}}\}\!\} &:= \tfrac{1}{2}(\underline{\mathbf{v}}^+ + \underline{\mathbf{v}}^-) \quad \text{on } \Gamma_I, \end{aligned}$$

and with the help of the outer product,$(\mathbf{a} \otimes \mathbf{b})_{ij} := a_i b_j$, a jump operator is defined as

$$\underline{[\![\mathbf{v}]\!]} := \mathbf{v}^+ \otimes \mathbf{n}^+ + \mathbf{v}^- \otimes \mathbf{n}^- \quad \text{on } \Gamma_I.$$

Here, $\mathbf{v}^+$ and $\mathbf{v}^-$ on the interior face $f = \partial\kappa^+ \cap \delta\kappa^- \in \Gamma_I$ denote the traces of $\mathbf{v}$ on κ^+ and κ^-. Recall, that equation (3.3) is written as a sum over all $\kappa \in \mathcal{T}_h$ as follows

$$\sum_{\kappa \in \mathcal{T}_h} \int_\kappa (\nabla \cdot (\mathcal{F}^c(\mathbf{u}) - \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u})) - \mathcal{S}(\mathbf{u}, \nabla\mathbf{u})) \cdot \mathbf{v} \, \mathrm{d}V = 0, \, \forall \, \mathbf{v} \in H^{m,b}(\mathcal{T}_h). \tag{3.6}$$

Now, for every summand the Gauss theorem is applied.

$$\begin{aligned} &\int_\kappa -\mathcal{F}^c(\mathbf{u}) : \nabla\mathbf{v} + \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) : \nabla\mathbf{v} - \mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{v} \, \mathrm{d}V \\ &+ \int_{\partial\kappa} (\tilde{\mathcal{F}}^c(\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v} - (\tilde{\mathcal{F}}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v} \, \mathrm{d}s = 0, \end{aligned} \tag{3.7}$$

where the notation $\sigma : \tau = \sum_{k=1}^n \sum_{l=1}^m \sigma_{kl}\tau_{kl}$ is used, for matrices $\sigma, \, \tau \in \mathbb{R}^{n \times m}$. Note that the terms $(\tilde{\mathcal{F}}^c(\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v}$ and $(\tilde{\mathcal{F}}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v}$ on the faces $\partial\kappa$ are not yet defined.

Since $H^{m,b}(\mathcal{T}_h)$ allows discontinuities over the faces $f = \partial\kappa^+ \cap \delta\kappa^- \in \Gamma_I$, with possibly different values $\mathbf{u}^+$ and $\mathbf{u}^-$ the normal flux $\tilde{\mathcal{F}}^c(\mathbf{u}) \cdot \mathbf{n}$ is replaced by a numerical flux function $\mathcal{H}^{fl}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n})$. Here, numerical flux functions can be used like known from finite volume (FV) methods, see Section 3.2. Moreover, integrals over the boundary Γ can also be expressed with numerical flux functions as described in Section 3.3.

Hence, (3.7) can be rewritten as

$$\begin{aligned} &\int_\kappa -\mathcal{F}^c(\mathbf{u}) : \nabla\mathbf{v} + \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) : \nabla\mathbf{v} - \mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{v} \, \mathrm{d}V \\ &+ \int_{\partial\kappa} \mathcal{H}^{fl}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) \cdot \mathbf{v}^+ - (\tilde{\mathcal{F}}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v} \, \mathrm{d}s = 0. \end{aligned} \tag{3.8}$$

In order to replace the term $(\tilde{\mathcal{F}}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v}$ on $\partial\kappa$ the viscous flux is converted into a system of first order differential equations. Applying the scheme of Bassi and Rebay (BR2) [5, 10], $(\tilde{\mathcal{F}}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v}$ is replaced by

$$(\tilde{\mathcal{F}}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v} = \frac{1}{2}\underline{[\![\mathbf{u}]\!]} : \mathbf{G}^\top(\mathbf{u})\nabla\mathbf{v}^+ + \mathbf{v}^+ \otimes \mathbf{n}^+ : \{\!\{\mathbf{G}(\mathbf{u})\nabla\mathbf{u}\}\!\} - \mathbf{v}^+ \otimes \mathbf{n}^+ : \underline{\delta}(\mathbf{u}),$$

where the notation is used as introduced in Definition 3.4. Here the matrix entries, with the help of equations (2.14) & (2.15), are defined as

$$(\mathbf{G}(\mathbf{u})\nabla\mathbf{u})_{ij} = \sum_{s=1}^{d}\sum_{l=1}^{n}(\mathbb{G}_{js}(\mathbf{u}))_{il}\frac{\partial u_l}{\partial x_s} \qquad \text{and}$$

$$\left(\mathbf{G}^\top(\mathbf{u})\nabla\mathbf{v}\right)_{ij} = \sum_{s=1}^{d}\sum_{l=1}^{n}(\mathbb{G}_{sj}(\mathbf{u}))_{li}\frac{\partial v_l}{\partial x_s}, \quad \forall\ i \in \underline{d+m}, j \in \underline{d}.$$

Note, like in Definition 3.3, that $n = d + 3$ for the RANS-SA equations and $n = d + 4$ for the RANS-$k\omega$ equations. Moreover, the penalization term for the numerical discretization of the viscous terms $\mathbf{v}^+ \otimes \mathbf{n}^+ : \underline{\delta}(\mathbf{u})$, is given by the BR2 scheme [5] (cf. [28] for a detailed description of $\underline{\delta}(\mathbf{u})$).

y, (3.8) can be rewritten as

$$\begin{aligned}
&\int_\kappa -\mathcal{F}^c(\mathbf{u}) : \nabla\mathbf{v} + \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u}) : \nabla\mathbf{v} - \mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{v}\ \mathrm{d}V \\
&+ \int_{\partial\kappa} \mathcal{H}^{fl}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) \cdot \mathbf{v}^+\ \mathrm{d}s \\
&- \int_{\partial\kappa} \frac{1}{2}\underline{[\![\mathbf{u}]\!]} : \mathbf{G}^\top(\mathbf{u})\nabla\mathbf{v}^+ + \mathbf{v}^+ \otimes \mathbf{n}^+ : \{\!\{\mathbf{G}(\mathbf{u})\nabla\mathbf{u}\}\!\} - \mathbf{v}^+ \otimes \mathbf{n}^+ : \underline{\delta}(\mathbf{u})\ \mathrm{d}s = 0.
\end{aligned} \tag{3.9}$$

After summation over all elements $\kappa \in \mathcal{T}_h$, the DG discretization of equation (3.3) is obtained as follows:

$$\text{Find } \mathbf{u}_h \in \mathcal{V}_h^p(\mathcal{T}_h) \text{ such that } N_h(\mathbf{u}_h, \mathbf{v}_h) = 0\ \forall\ \mathbf{v}_h \in \mathcal{V}_h^p(\mathcal{T}_h), \tag{3.10}$$

where $N_h(\mathbf{u}_h, \mathbf{v}_h)$ is given by

$$\begin{aligned}
N_h(\mathbf{u}_h, \mathbf{v}_h) = \ &\int_{\mathcal{T}_h} (-\mathcal{F}^c(\mathbf{u}_h) + \mathcal{F}^v(\mathbf{u}_h, \nabla\mathbf{u}_h)) : \nabla\mathbf{v}_h - \mathcal{S}(\mathbf{u}_h, \nabla\mathbf{u}_h) \cdot \mathbf{v}_h\ \mathrm{d}V \\
&- \int_{\Gamma_I \cup \Gamma} \underline{[\![\mathbf{u}_h]\!]} : \{\!\{\mathbf{G}^\top(\mathbf{u}_h)\nabla\mathbf{v}_h\}\!\} + \underline{[\![\mathbf{v}_h]\!]} : \{\!\{\mathbf{G}(\mathbf{u}_h)\nabla\mathbf{u}_h\}\!\}\ \mathrm{d}s \\
&+ \sum_{\kappa\in\mathcal{T}_h} \int_{\partial\kappa} \mathcal{H}^{fl}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n}) \cdot \mathbf{v}_h^+\ \mathrm{d}s + \int_{\Gamma_I \cup \Gamma} \underline{[\![\mathbf{v}_h]\!]} : \underline{\delta}(\mathbf{u}_h)\ \mathrm{d}s.
\end{aligned} \tag{3.11}$$

For a detailed description of the discretization on Γ see Section 3.3, since the jump operators from Definition 3.4, which originate from the BR2 scheme in (3.11), are not yet defined on Γ. Moreover, the convective numerical flux function occurring in the face integrals in (3.11) are defined in Section 3.2.

3.2 Numerical flux function for convective terms

Note that parts of this subsection are already published in [76]. The numerical flux formulations used in this work will be based on the eigen-decomposition of the normal convective flux Jacobian,

$$\mathcal{F}_n^{c\prime}(\mathbf{u}) := \frac{\partial(\mathcal{F}^c(\mathbf{u})\cdot\mathbf{n})}{\partial\mathbf{u}}. \tag{3.12}$$

Explicit expressions for the resulting 3D Jacobian matrix for the RANS-$k\omega$ and RANS-SA equations can be found in Appendix B. This Jacobian can be written as

$$\mathcal{F}_n^{c\prime} = T\Lambda T^{-1},$$

where $\Lambda = \mathrm{diag}(\lambda_i)$ is the diagonal matrix of eigenvalues of $\mathcal{F}_n^{c\prime}$ and $T = (\mathbf{w}_1, \ldots, \mathbf{w}_7)$ is the matrix of eigenvectors $\mathbf{w}_i$ corresponding to the eigenvalues λ_i. This eigen-decomposition corresponds to the transformation to a set of characteristic variables, like described in Section 2.2, which can be treated via upwinding.

Moreover, the numerical flux functions presented in this section are consistent and conservative, i.e.

- $\mathcal{H}^{\mathsf{fl}}(\cdot,\cdot,\cdot)$ is consistent if
$$\mathcal{H}^{\mathsf{fl}}(\mathbf{u},\mathbf{u},\mathbf{n})|_{\partial\kappa} = \mathcal{F}^c(\mathbf{u})\cdot\mathbf{n}, \tag{3.13}$$
- $\mathcal{H}^{\mathsf{fl}}(\cdot,\cdot,\cdot)$ is conservative if
$$\mathcal{H}^{\mathsf{fl}}(\mathbf{v},\mathbf{u},\mathbf{n}) = -\mathcal{H}^{\mathsf{fl}}(\mathbf{u},\mathbf{v},-\mathbf{n}). \tag{3.14}$$

3.2.1 Roe flux

The Roe flux [59] can be regarded as an average of the fluxes evaluated for the left and right state plus a dissipation stabilization of the discontinuities applied to the conservative variables. It is given by

$$\mathcal{H}^{\mathsf{Roe}}(\mathbf{u}^+,\mathbf{u}^-,\mathbf{n}) = \frac{1}{2}\left(\mathcal{F}^c(\mathbf{u}^+)\cdot\mathbf{n} + \mathcal{F}^c(\mathbf{u}^-)\cdot\mathbf{n} + T\,A\,T^{-1}(\mathbf{u}^+ - \mathbf{u}^-)\right) \tag{3.15}$$

with $A = \mathrm{diag}(\alpha_i)$ and $\alpha_i = |\lambda_i|$ in the original Roe scheme. However, this introduces no dissipation for characteristic variables whose eigenvalues are zero. Several options for an *entropy fix* have been proposed to overcome this problem. A rather simple technique is given by choosing $\alpha_i = \max(|\lambda_i|, \delta_{\mathsf{ef}}\lambda_{\mathsf{max}})$, where $\lambda_{\mathsf{max}} = \max_i |\lambda_i|$ and $0 \leq \delta_{\mathsf{ef}} \leq 1$ is a free parameter, typically $\delta_{\mathsf{ef}} = 0.1$ [47]. The original scheme without entropy fix is recovered by

$\delta_{\text{ef}} = 0$, whereas $\delta_{\text{ef}} = 1$ yields the local Lax-Friedrichs [42] or Rusanov [62] flux $\mathcal{H}^{\text{LF}}$, which can be written in simpler form as

$$\mathcal{H}^{\text{LF}}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) = \frac{1}{2}\left(\mathcal{F}^c(\mathbf{u}^+)\cdot\mathbf{n} + \mathcal{F}^c(\mathbf{u}^-)\cdot\mathbf{n} + \lambda_{\text{max}}(\mathbf{u}^+ - \mathbf{u}^-)\right). \tag{3.16}$$

A second choice for the entropy fix is inspired by Harten [25],

$$\alpha_i = \begin{cases} \frac{|\lambda_i|^2+\delta^2}{2\delta} & \text{if } |\lambda_i| < \delta, \\ |\lambda_i| & \text{else,} \end{cases} \tag{3.17}$$

with $\delta = \delta_{\text{ef}}\lambda_{\text{max}}$ and $0 \leq \delta_{\text{ef}} < 1$. This formulation has the advantage of being differentiable at the point $|\lambda_i| = \delta$.

The linearization involved in the determination of T and λ_i is performed at a Roe-averaged state, which is given by $\bar{\rho} = \sqrt{\rho^+\rho^-}$ and $\bar{x} = \frac{\sqrt{\rho^+}x^+ + \sqrt{\rho^-}x^-}{\sqrt{\rho^+}+\sqrt{\rho^-}}$ for $x = v_i, k, \omega, H$, where $H = E + \frac{p}{\rho}$ denotes the total enthalpy, as seen in (**??**) of Chapter 2.

Note, that all computations in this work are based on the Roe flux and the Harten's entropy fix with $\delta_{\text{ef}} = 0.2$.

3.2.2 Eigenvalue decomposition of the flux Jacobian

In this section the eigenvalues and eigenvectors of the Jacobian for the RANS-$k\omega$ and RANS-SA equations are given in 3D. Results for 2D are very similar and, therefore, not shown here, for brevity.

Eigenvalue decomposition for the RANS-$k\omega$ equations

The eigenvalues of the normal flux Jacobian $\mathcal{F}_n^{c\prime}$ in (3.12) are $\mathbf{v}\cdot\mathbf{n}$ in multiplicity 5, where $\mathbf{v} = (v_1, \ldots, v_d)^\top$ is the velocity vector and $\mathbf{n}$ the normal vector. The missing two eigenvalues are $\mathbf{v}\cdot\mathbf{n} \pm a_t$ with a turbulent speed of sound $a_t = \sqrt{a^2 + \frac{2}{3}\gamma k} = \sqrt{\gamma\frac{p_{\text{eff}}}{\rho}}$ based on the effective pressure defined in analogy to the usual speed of sound $a = \sqrt{\gamma\frac{p}{\rho}}$ based on the thermodynamic pressure. Similar eigenvalues for the Jacobian of the steady-state Navier-Stokes equations where stated in (2.10). The only difference is the use of the usual speed of sound a in (2.10).

Using the Gaussian algorithm, the first five eigenvectors of $\mathcal{F}_n^{c\prime}$ are calculated as a kernel of $\mathcal{F}_n^{c\prime} - \mathbf{v}\cdot\mathbf{n}\,\mathbf{I}$:

$$\mathbf{w}_1 := \left(n_1, n_1v_1, n_1v_2 + n_3a_t, n_1v_3 - n_2a_t, \tfrac{n_1}{2}\mathbf{v}^2 + a_t(n_3v_2 - n_2v_3), 0, 0\right)^\top,$$

$$\mathbf{w}_2 := \left(n_2, n_2v_1 - n_3a_t, n_2v_2, n_2v_3 + n_1a_t, \tfrac{n_2}{2}\mathbf{v}^2 + a_t(n_1v_3 - n_3v_1), 0, 0\right)^\top,$$

$$\mathbf{w}_3 := \Big(n_3, n_3 v_1 + n_2 a_t, n_3 v_2 - n_1 a_t, n_3 v_3, \tfrac{n_3}{2}\mathbf{v}^2 + a_t(n_2 v_1 - n_1 v_2), 0, 0\Big)^\top,$$

$$\mathbf{w}_4 := \Big(0, 0, 0, 0, \gamma - \tfrac{5}{3}, \gamma - 1, 0\Big)^\top \text{ and } \mathbf{w}_5 := \Big(0, 0, 0, 0, 0, 0, 1\Big)^\top.$$

Here, $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$ are analogous to the results for the Euler equations without turbulence model [39], with the speed of sound replaced by a_t, and two additional zero components for the turbulence model variables.

However, above calculation fails for the remaining two eigenvectors because of coefficient growth over the field $K := \mathbb{Q}(n_1, n_2, n_3, v_1, v_2, v_3, E, k, \gamma, \omega, a_t)$. Instead, using the GINV-system[12] the sixth eigenvalue is computed symbolically as the kernel of $\mathcal{F}_n^{c\prime} - (a_t + \mathbf{v}\cdot\mathbf{n})\,\mathbf{I}$ over the ring $R := \mathbb{Q}[n_1, n_2, n_3, v_1, v_2, v_3, E, k, \gamma, \omega, a_t]$:

$$\mathbf{w}_6 := \Big(1, v_1 + a_t\, n_1, v_2 + a_t\, n_2, v_3 + a_t\, n_3, E + \tfrac{p + \frac{2}{3}\rho k}{\rho} + a_t\, \mathbf{v}\cdot\mathbf{n}, k, \omega\Big)^\top.$$

This non-zero element $\mathbf{w}_6$ of the kernel of $\mathcal{F}_n^{c\prime} - (a_t + \mathbf{v}\cdot\mathbf{n})\,\mathbf{I}$ over R is a non-zero element in the kernel over the quotient field K of R. The latter kernel over K is one-dimensional, so $\mathbf{w}_6$, generates this kernel. By substituting a_t by $-a_t$ in $\mathbf{w}_6$ the seventh eigenvector is obtained,

$$\mathbf{w}_7 := \Big(1, v_1 - a_t\, n_1, v_2 - a_t\, n_2, v_3 - a_t\, n_3, E + \tfrac{p + \frac{2}{3}\rho k}{\rho} - a_t\, \mathbf{v}\cdot\mathbf{n}, k, \omega\Big)^\top.$$

Eigenvalue decomposition for the RANS-SA equations

The eigenvalues of the convective flux Jacobian $\mathcal{F}_n^{c\prime}$ from (3.12) are $\mathbf{v}\cdot\mathbf{n}$ in multiplicity 4 and $\mathbf{v}\cdot\mathbf{n} \pm a$ with the speed of sound $a = \sqrt{\gamma \frac{p}{\rho}}$ based on the thermodynamic pressure. The same eigenvalues where stated also for the flux Jacobian of the steady-state Navier-Stokes equations in (2.10). All eigenvectors of $\mathcal{F}_n^{c\prime}$ are calculated using the Gaussian algorithm:

$$\mathbf{w}_1 := \Big(n_1, n_1 v_1, n_1 v_2 + n_3 a, n_1 v_3 - n_2 a, \tfrac{n_1}{2}\mathbf{v}^2 + a(n_3 v_2 - n_2 v_3), 0\Big)^\top,$$

$$\mathbf{w}_2 := \Big(n_2, n_2 v_1 - n_3 a, n_2 v_2, n_2 v_3 + n_1 a, \tfrac{n_2}{2}\mathbf{v}^2 + a(n_1 v_3 - n_3 v_1), 0\Big)^\top,$$

$$\mathbf{w}_3 := \Big(n_3, n_3 v_1 + n_2 a, n_3 v_2 - n_1 a, n_3 v_3, \tfrac{n_3}{2}\mathbf{v}^2 + a(n_2 v_1 - n_1 v_2), 0\Big)^\top,$$

$$\text{and } \mathbf{w}_4 := \Big(0, 0, 0, 0, 0, 1\Big)^\top.$$

Here, $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$ are analogous to the results shown for the RANS-$k\omega$ equations with only one additional zero component for the turbulence model variables. By replacing a_t with a in $\mathbf{w}_6$ the RANS-$k\omega$ equations, and setting $k = 0$, the fifth eigenvector is obtained,

$$\mathbf{w}_5 := \Big(1, v_1 + a\, n_1, v_2 + a\, n_2, v_3 + a\, n_3, H + a\, \mathbf{v}\cdot\mathbf{n}, \nu_t\Big)^\top.$$

By substituting a by $-a$ in $\mathbf{w}_5$ the last eigenvector is obtained,

$$\mathbf{w}_6 := \Big(1, v_1 - a\,n_1, v_2 - a\,n_2, v_3 - a\,n_3, H - a\,\mathbf{v}\cdot\mathbf{n}, \nu_t\Big)^\top.$$

3.3 Boundary conditions

In this section, all boundary conditions prescribed on the boundary Γ of the computational domain Ω are addressed. Considered are different types of boundaries, namely the wall and the far field boundaries. The convective numerical flux functions, introduced in Section 3.2 on interior faces, are also applied to faces at the boundary Γ. In analogy, the jump and mean value operators from Definition 3.4 forming the diffusive numerical flux on interior faces are also applied to boundary faces. In fact, the main idea is to evaluate the discretization on the boundaries in the same way as on interior faces. These boundary conditions are consistent and also do not violate the adjoint consistency of the discretization, see [28].

At Γ, no exterior element κ^- is available for providing the exterior state $\mathbf{u}^-$ required in the evaluation of the numerical flux function $\mathcal{H}^{\text{fl}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n})$ or jump and mean value operators. Therefore, an exterior boundary state $\mathbf{u}_\Gamma^-(\mathbf{u}_h^+)$ on the boundary $\partial\kappa^+ \cap \Gamma$ is computed depending on the interior state $\mathbf{u}_h^+$ based on the idea of introducing a ghost layer of elements at the boundaries.

Wall boundary condition

At the wall boundary, a numerical flux is evaluated connecting the interior state $\mathbf{u}_h^+$ with a mirrored exterior boundary state $\mathbf{u}_\Gamma^-$. Here, $\mathbf{u}_\Gamma^-$ denotes a boundary exterior state which depends on the boundary state $\mathbf{u}_\Gamma(\mathbf{u}_h^+)$ but which also depends on the interior state $\mathbf{u}_h^+$, and is given by

$$\frac{1}{2}(\mathbf{u}^+ + \mathbf{u}_\Gamma^-(\mathbf{u}_h^+)) = \mathbf{u}_\Gamma(\mathbf{u}_h^+). \tag{3.18}$$

With that in mind, the Roe flux with the outer state $\mathbf{u}_\Gamma^-(\mathbf{u}_h^+)$ is employed as $\mathcal{H}^{\text{Roe}}(\mathbf{u}_h^+, \mathbf{u}_\Gamma^-, \mathbf{n})$.

As an addition to Definition 3.4, the following definition on the boundary Γ is stated.

Definition 3.5:

Let Γ be the boundary of Ω. Consider a vector-valued function $\mathbf{v} \in H^{m,b}(\mathcal{T}_h)$, a matrix-valued function $\underline{\mathbf{v}} \in [H^{m,b}(\mathcal{T}_h)]^d$ and $\mathbf{n}^+$ the unit outward normal vector κ^+.

Then the mean values at the boundary are defined by

$$\begin{aligned} \{\!\{\mathbf{v}\}\!\} &:= \tfrac{1}{2}(\mathbf{v}^+ + \mathbf{v}_\Gamma^-(\mathbf{v}^+)) \quad \text{on } \Gamma, \\ \{\!\{\underline{\mathbf{v}}\}\!\} &:= \tfrac{1}{2}(\underline{\mathbf{v}}^+ + \underline{\mathbf{v}}_\Gamma^-(\mathbf{v}^+)) \quad \text{on } \Gamma, \end{aligned}$$

and with the help of the outer product the last jump operator is defined as

$$\underline{[\![\mathbf{v}]\!]} := \mathbf{v}^+ \otimes \mathbf{n}^+ + \mathbf{v}_\Gamma^-(\mathbf{v}^+) \otimes (-\mathbf{n}^+) \quad \text{on } \Gamma.$$

Moreover, the penalty function $\underline{\delta}(\mathbf{u}_h)$ of the BR2 scheme [5] on Γ is handled based in the same basic idea and is described in [28].

Finally, the wall exterior gradient $(\nabla\mathbf{u})_\Gamma^-$ required in the viscous numerical flux at the boundary is defined to be identical to the interior gradient, i.e. $(\nabla\mathbf{u})_\Gamma^- := \nabla\mathbf{u}_h^+$.

Please recall that there is still an issue with the wall boundary condition of $\tilde{\omega}$ at Γ_{wall} for the discretization of the Reynolds-Averaged Navier-Stokes (RANS)-$k\omega$ equations, since the analytical solution on the wall is infinite. Hence, the wall boundary condition proposed in [64] is applied in this work. This sets $\tilde{\omega}_{\text{wall}}$, from Section 2.3.1, and, therefore, the entry for $\tilde{\omega}$ in $\mathbf{u}_\Gamma(\mathbf{u}_h^+)$ at the wall to a polynomial dependent and finite but very large value.

Then the boundary state at the wall for the RANS-$k\omega$ and RANS-SA equations is defined as

$$\mathbf{u}_\Gamma^{k\omega}(\mathbf{u}_h^+) = (\rho^+, 0, \ldots, 0, \rho^+E^+, 0, \rho^+\tilde{\omega}_{\text{wall}}^+)^\top, \tag{3.19}$$

$$\mathbf{u}_\Gamma^{SA}(\mathbf{u}_h^+) = (\rho^+, 0, \ldots, 0, \rho E^+, 0)^\top. \tag{3.20}$$

Characteristic far field boundary condition

At the outer boundaries of the computational domain, far field boundary conditions are applied in the same way as described above. A far field state $\mathbf{u}^\infty$ is defined as the state of the free undisturbed flow. The number of prescribed values at each point on the in- and outflow boundaries should correspond to the number of incoming and outgoing characteristics in each case. To this end, far field boundary conditions are prescribed in a weak form employing the Roe flux [59] with the outer state $\mathbf{u}_h^-$ replaced by $\mathbf{u}^\infty$. Following the definitions from Chapter 2, the freestream values $\mathbf{u}^\infty$ for the RANS-$k\omega$ and RANS-SA equations are given by,

$$\mathbf{u}_\infty^{k\omega} = \left(\rho_\infty, \rho_\infty v_{1,\infty}, \ldots, \rho_\infty v_{d,\infty}, \rho_\infty E_\infty, \rho_\infty k_\infty, \rho_\infty \tilde{\omega}_\infty\right)^\top, \tag{3.21}$$

$$\mathbf{u}_\infty^{SA} = \left(\rho_\infty, \rho_\infty v_{1,\infty}, \ldots, \rho_\infty v_{d,\infty}, \rho_\infty E_\infty, \rho_\infty \tilde{\nu}_\infty\right)^\top. \tag{3.22}$$

This technique is also referred to as a characteristic far field boundary condition.

3.4 Basis functions

The key aspect in this section is the definition of basis functions directly in physical space, which is also referred to as a non-parametric formulation. Furthermore, the space is equivalent to the choice of a Taylor basis [45]. This way, the local basis can be defined independently from a specific reference element, which is required for the agglomeration multigrid algorithms in Section 4. Moreover, an ortho-normal hierarchic basis formulation is used in order to simplify the transfer operator formulation, see Section 4.6. The choice of such a space in connection with the formulation of a DG method on agglomerated meshes has been described before in [3, 4].

For moderately degrees ($p = 0 - 2$) the polynomial basis is obtained in the following way. Starting point on each mesh element is the monomial basis, e.g. for $p = 2$ in 2D it is $(1, x, y, xy, x^2, y^2)$. As can be seen for the given example, every polynomial of a degree up to 2 can be constructed as a linear combination of the stated monomials; Therefore, it is indeed a basis. The dimension of this basis is six for $p = 2$, for any $p \geq 0$ the dimension $d_{p,2D}$ of such a constructed basis in 2D is given by

$$d_{p,2D} = \sum_{i=0}^{p} (i + 1), \tag{3.23}$$

whereas in 3D it is given by

$$d_{p,3D} = \sum_{j=0}^{p} \sum_{i=0}^{j} (i + 1). \tag{3.24}$$

The monomial basis is ortho-normalized with the L^2 scalar product on every mesh element via the modified Gram–Schmidt process and results in a mass matrix which equals the identity matrix.

Our observation is that, for higher degree polynomials on a mesh with stretched elements and curved faces, instabilities in the flow computation will appear. This occurs because the mass matrices of such an element may become unacceptably different from the identity matrix, as seen in [3]. To avoid such an unwanted behavior, the same alterations of the basis functions are performed as in [3, 4]. These changes are in such a way that the basis functions are defined in a coordinate system which is shifted with the help of the element midpoint and rotated along the local element frame to account for the geometric anisotropy of the cell. This alternation stabilizes the DG discretization of the RANS equations on higher order turbulent meshes and polynomials of a degree greater than two. For computations using polynomials with a degree smaller than three no noteworthy difference between both formulations was observed in this work. Moreover, recent

results indicate that in this area of application the most gains in terms of accuracy over computational effort are gained by approaches up to $p = 2$ or $p = 3$, anyway [77].

Furthermore, the practical implementation of the method requires a way to evaluate integrals via a numerical quadrature with high order of accuracy. This task is done via Gaussian quadrature rules. These are available for standard reference elements which can be mapped to real mesh elements, but there is no general formulation on polyhedra. Thus, the integration over agglomerated elements has to be split into integrals over subregions which can be mapped to a given set of reference elements. To this end, general polyhedra may be triangulated. For agglomerated meshes there is a simpler choice, however. Avoiding the task of performing an additional triangulation for each agglomerate the integration can be performed over the elements of the underlying fine mesh. This technique results in a large number of quadrature points on each agglomerate and thus a high computational cost for each operator evaluation. While this is the only way to ensure exact integration of polynomial functions of a given degree, the cost might be reduced by considering the fact that the absolute accuracy of integration can be rather good using lower order integration formulae over the large number of sub-regions defined by the fine mesh. This approach has also been suggested before [3, 4], but it is not applied here. This type of run-time behavior optimization is considered to be part of future research, and concentrate on the capabilities of the basic algorithm in the current work. Therefore, the integration on the agglomerated meshes will be based on all quadrature points of the original mesh. Hence, for h-multigrid methods the scalability is not optimal for the integration on agglomerated elements.

3.5 Mesh related issues

This section gives a basic introduction in mesh related issues for higher order FE methods for turbulent flows.

The use of highly stretched meshes for an optimal resolution of turbulent boundary layers is essential to solve the RANS equation in combination with a turbulence model accurately. This is true for all mesh dependent methods like, e.g. FV and FE methods.

In higher order DG methods it is essential, to approximate curved wall boundaries with polynomials of a degree greater than one [8, 9]. Using a boundary representation with polynomials of degree one, corresponds to a standard FV boundary representation. For higher order DG methods, this representation would lead to kinks in the boundary representation and would trigger a physical but unwanted rarefaction or shock wave behind every kink, which would ruin the higher order flow solution [8, 9].

In most cases, meshing tools do not support the creation of higher order meshes with curved boundaries. One meshing tool which supports higher order unstructured mesh making is GMSH [24]. Note that results on meshes created with GMSH are shown in this work, one of which can be seen in Figure 3.1. Creating a higher order mesh with the help of a meshing tool works reasonable well in 2D without a deeper knowledge in this field. In 3D, however, the direct higher order mesh generation with a given meshing tool is not near industrial use. Therefore, two alternative approaches to create higher order meshes are introduced below.

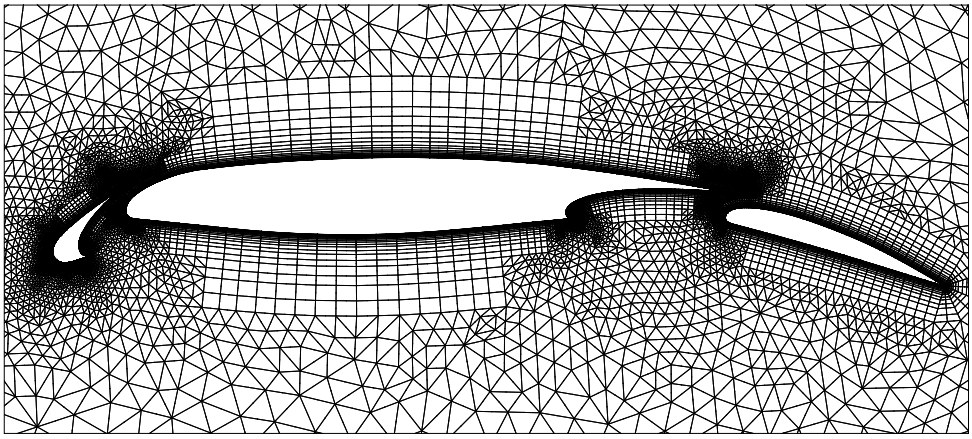

Figure 3.1: Unstructured higher order L1T2 mesh created via GMSH.

Moreover, it is rather difficult to generate meshes that are very coarse globally but still feature a reasonable resolution of both geometric and flow features. Thus, unstructured meshes tend to be finer than the coarsest structured ones. Due to the increased mesh size, they are usually treated with moderately high order in the DG context. Please recall that results indicate that in this area of application the most gains in terms of computational effort per accuracy are gained by approaches up to third or fourth order, anyway [77].

In the past, some approaches have been developed to create higher order meshes without the use of a higher order meshing tool. The most common one relies on a structured FV mesh which is coarsened in a way that the coarsened mesh is structured as well. The unused points from the initial mesh are used for the polynomial approximation of the boundaries in the coarsened mesh. Note that all edges of mesh elements, even if they are not at a boundary, will be approximated with polynomials with the same technique, see Figure 3.2, inorder to avoid negative volumes in the boundary layer.

In order to represent the boundary with higher polynomial degrees, the fine mesh needs to be coarsened more often. This is due to the fact that a polynomial of degree n can be represented in 2D uniquely with $n + 1$ points and with more coarsening steps more points are free to assist with the boundary representation. For example, in 2D, one point is free after one coarsening step, after two steps three free points and after three steps seven free points are available. These free points, together with the two points which are

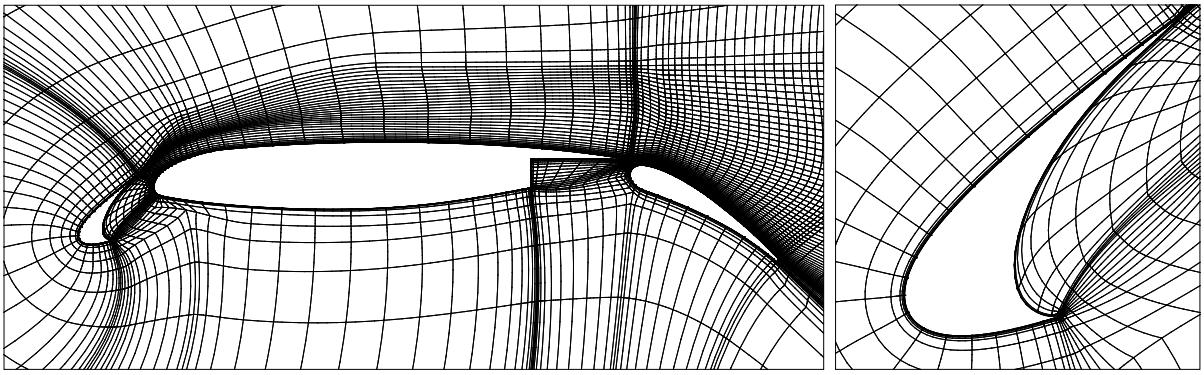

Figure 3.2: Structured higher order MDA 30P30N mesh created via coarsening and additional points: full geometry (left), slate zoom (right).

still connected to the beginning and end of an edge, can be used to create a polynomial representation of the boundary. Details of a mesh which was created via this approach are shown in Figure 3.2. The curved element edges are approximated via polynomials of degree four. Hence, at least two coarsening steps of the initial finite volume mesh were performed in order to have five points for a proper representation. As can be seen at the front tip of the slate in Figure 3.2 (right) the elements are very stretched in the boundary layer, however, the geometry of the slate is represented accurately. Please note that even if only 2D examples are given, this technique is also applicable in 3D.

One other approach relies on the linear elasticity equation based on the analogy of a solid body under deformation for a generic elastic material [15, 27]. Here, the standard linear mesh is curved by using the original underlying boundary representation in combination with the linear elasticity equation. The basic idea is to allow polynomials as mesh edges and curve them along the original boundary representation and propagate this curvature into the mesh near the boundary in order to avoid negative cell volumes.

4 Solver methods

In this chapter the solver algorithms are given to solve the discretized RANS equations. A brief overview of the Chapters 2-4 is given in Figure 4.1. Moreover, basic definitions are given for the solver algorithms applied in this work. The top blue bubbles in Figure 4.1 represent the main equations this work focuses on which are described in Chapter 2. The Navier-Stokes equations are nonlinear partial differential equations (PDEs), see (2.1). After a mass-weighted time-averaging of the Navier-Stokes equations the Reynolds-averaged Navier-Stokes (RANS) equations are derived [16]. The resulting nonlinear stationary RANS equations are combined with a turbulence model, see Figure 4.1. These RANS equations from (2.18) in combination with a turbulence model are discretized with the help of discontinuous Galerkin (DG) methods, see the red bubbles in Figure 4.1. The DG discretization of the weak formulation of equation (2.18) was obtained in (3.10) as:

$$\text{Find } \mathbf{u}_h \in \mathcal{V}_h^p(\mathcal{T}_h) \text{ such that } N_h(\mathbf{u}_h, \mathbf{v}_h) = 0 \ \forall \ \mathbf{v}_h \in \mathcal{V}_h^p(\mathcal{T}_h), \tag{4.1}$$

where $N_h(\mathbf{u}_h, \mathbf{v}_h)$ as in (3.11) and the numerical representation of the integrals is accomplished via Gauss quadrature on the given mesh [36]. This DG discretization is represented on a computational mesh $\mathcal{T}_h$ with a polynomial basis of $\mathcal{V}_h^p(\mathcal{T}_h)$, as described in Section 3.4. Since the number of basis functions is finite and the mesh has a finite number of elements the resulting system of nonlinear equations from (4.1) is finite as well as described in Chapter 3.

Let $\mathbf{L}(\mathbf{u}) = \mathbf{f}$ be this discretized nonlinear problem, with $\mathbf{f} = 0$. Then the nonlinear residual is defined as $\mathbf{R}_{l_{\text{max}}}(\mathbf{u}_{l_{\text{max}}}) := -\mathbf{L}(\mathbf{u})$. This nonlinear residual needs to be reduced in order to solve the nonlinear problem. This will be achieved with the help of solver methods presented in this chapter, see the green bubbles in Figure 4.1, resulting in numerical solutions of the RANS equations for several numerical meshes and DG discretizations. First, the nonlinear solvers are introduced, then the linear solvers are presented and finally the whole solver framework is described and analyzed.

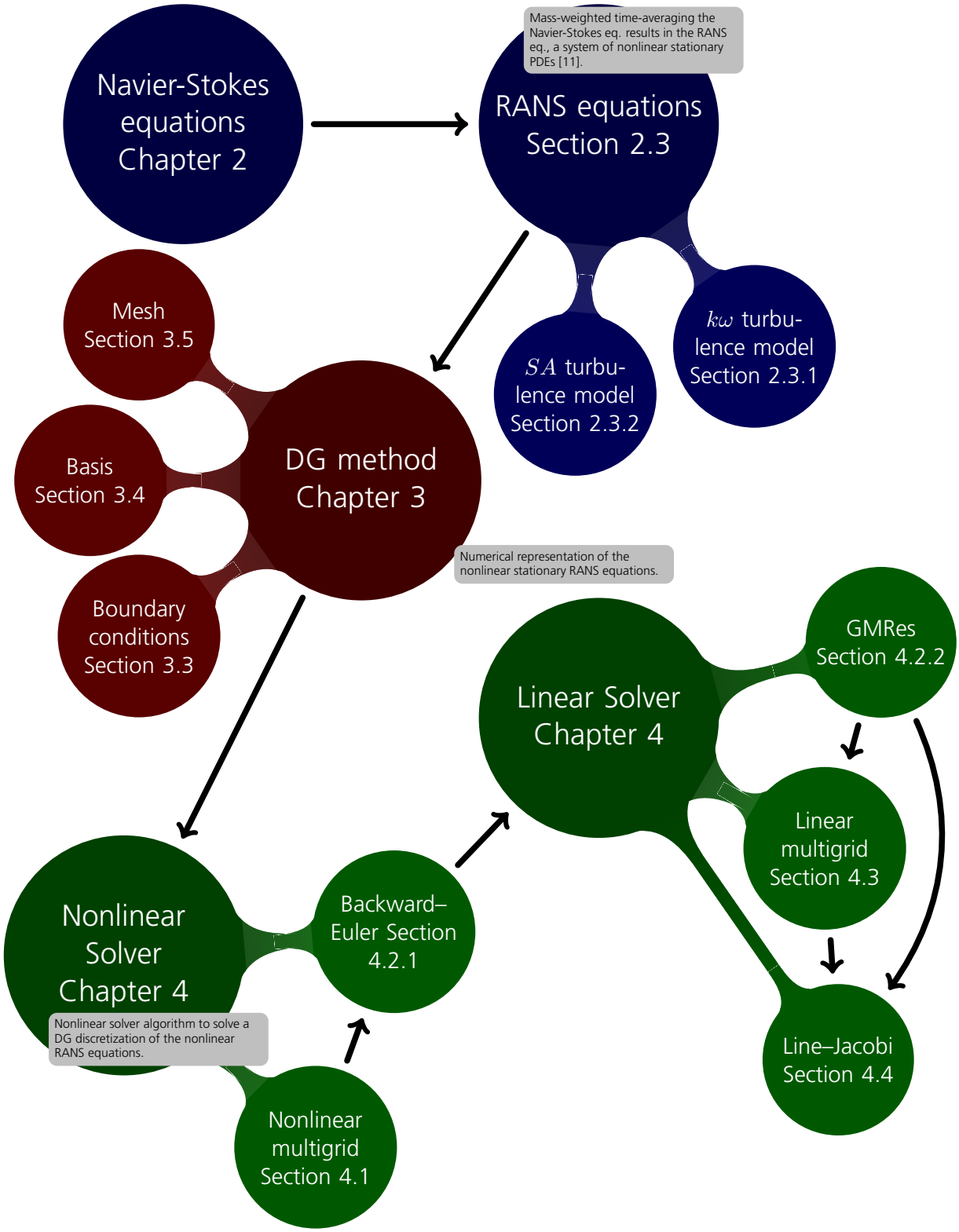

Figure 4.1: Overview of the structure of Chapters 2-4.

In this chapter several strategies are presented to exploit hierarchies of coarse level problems for the efficient solution of the discretized equations (2.18), including both level sequencing and linear as well as nonlinear multigrid variations [70]. Based on either lower order discretizations or agglomerated coarse meshes, the resulting algorithms can be characterized as either p- or h-multigrid algorithm, respectively. For convenience both kinds of algorithms are denoted as multigrid algorithms although the p-multigrid algorithm always uses the same computational mesh and would correctly be described as a multilevel algorithm. However, in the DG community the same notation is used [6, 7, 44, 76]. The only difference between these multigrid algorithms is the use of different coarse level DG discretizations and, therefore, transfer operators. All other ingredients like smoothers, timestep control, usage of a Galerkin transfer [70], startup strategy, etc. will stay the same for both kinds of multigrid algorithms.

A Rosenbrock method [60], in combination with an iterative linear solver, is considered a standard approach to solve a nonlinear set of equations in the DG community [5, 14, 26]. Only the Jacobian of the function, that is the Jacobian of the discretized nonlinear residual, is required. Typically, preconditioned Krylov subspace methods are chosen as linear solvers [63], with the ILU-preconditioned GMRes being the most common variant. This represents a well proven technique, close to a black-box approach. In contrast to that, multigrid-based algorithms aim at exploiting additional knowledge concerning the discrete equations, namely their origin as discretized partial differential equations (PDEs) using a given discretization method (including the polynomial degree in the DG case) on a given mesh, in order to enhance the behavior of the overall solution algorithm. The common idea of all multigrid methods is to construct additional coarse level problems which are used in a recursive correction approach.

For a p-multigrid method, polynomial linear spaces of different polynomial degree on the same computational mesh are chosen. For an h-multigrid method, the lower levels consist of coarse meshes, which are constructed from the computational mesh on the highest level. Whether it is a linear or a nonlinear multigrid method, the main idea will stay the same. A linear multigrid algorithm is characterized as a linear correction scheme since the state vector on a given level is corrected with approximations from lower levels [70]. Instead a nonlinear multigrid algorithm is typically characterized as a full approximation scheme since also the nonlinear state vector is transferred to the lower levels of the nonlinear multigrid algorithm [70]. The h- and p-multigrid methods are both defined by a nested hierarchy of linear spaces:

$$\mathbf{V}_{l_{\text{min}}} \subset \mathbf{V}_{l_{\text{min}}+1} \subset \cdots \subset \mathbf{V}_{l-1} \subset \mathbf{V}_l.$$

Please note, that from here on $\mathbf{V}_l$ describes the broken Sobolev spaces $\mathcal{V}_h^p(\mathcal{T}_h)$ on level l, which were introduced in Section 3.1. In a p-multigrid method, the polynomial degree p

in $\mathcal{V}_h^p(\mathcal{T}_h)$ would differ for different levels l, whereas for h-multigrid methods different $\mathcal{T}_h$ are used.

4.1 Nonlinear multigrid methods

In this section, a nonlinear multigrid algorithm is introduced and Chapter 5 gives a numerical analysis of the proposed algorithm.

Let $\mathbf{L}_l(\mathbf{u}_l) = \mathbf{f}_l$ be the nonlinear problem on level l.

Algorithm 1 Nonlinear Multigrid Algorithm $NMG(\mathbf{f_l}, \mathbf{u}_l, \mathbf{m}_1, \mathbf{m}_2, l, l_{\text{min}}, \tau)$

1: **if** $l = l_{\text{min}}$ **then**
2: $\quad \tilde{\mathbf{u}}_{l_{\text{min}}} := BWE_{l_{\text{min}}}(\mathbf{u}_{l_{\text{min}}}, m_{2,l_{\text{min}}})$
3: **if** $l > l_{\text{min}}$ **then**
4: $\quad \mathbf{u}_l := BWE_l(\mathbf{u}_l, m_{1,l})$ /* pre-smoothing */
5: $\quad \mathbf{u}_{l-1}^0 := \hat{I}_l^{l-1}\mathbf{u}_l$
6: $\quad \mathbf{f}_{l-1} := I_l^{l-1}\left(\mathbf{f}_l - \mathbf{L}_l(\mathbf{u}_l)\right) + \mathbf{L}_{l-1}(\mathbf{u}_{l-1}^0)$
7: $\quad$ **for** $k = 1$ **to** τ **do**
8: $\quad\quad \mathbf{u}_{l-1}^k := NMG(\mathbf{f}_{l-1}, \mathbf{u}_{l-1}^{k-1}, \mathbf{m}_1, \mathbf{m}_2, l-1, l_{\text{min}}, \tau)$
9: $\quad \tilde{\mathbf{u}}_l := \mathbf{u}_l + \hat{I}_{l-1}^l\left(\mathbf{u}_{l-1}^\tau - \mathbf{u}_{l-1}^0\right)$
10: $\quad \tilde{\mathbf{u}}_l := BWE_l(\tilde{\mathbf{u}}_l, m_{2,l})$ /* post-smoothing */
11: **return** $\tilde{\mathbf{u}}_l$

The nonlinear multigrid algorithm, shown in **Algorithm 1**, is defined recursively. Here, l denotes the current level and l_{min} the lowest level of the multigrid algorithm. Furthermore, $\mathbf{m}_1$ and $\mathbf{m}_2$ represent arrays with entries for every level, which hold the number of smoother steps $m_{1,l}$ and $m_{2,l}$ separately for each level l. For $l > l_{\text{min}}$, the state vector $\mathbf{u}_l$ is smoothed $m_{1,l}$-times with the help of the smoother algorithm $BWE_l(\cdot,\cdot)$ in line 4 of **Algorithm 1**. After that, the state vector and the defect between the various approximation levels are restricted to the space $\mathbf{V}_{l-1}$ in line 5-6. Please note that the restriction operators $\hat{I}_l^{l-1}$ and I_l^{l-1} are defined in a later stage of this chapter. In line 7-8, the recursion takes place with adapted values for the next level $l-1$. For $\tau = 1$ a V-cycle is performed, whereas $\tau = 2$ produces a W-cycle, see Figure 4.2. The difference between the iterated state vector from the lower levels and the restricted state vector from the beginning is prolongated with $\hat{I}_{l-1}^l$ and added to the state vector $\mathbf{u}_l$ on level l in line 9. At last in line 10, a second smoothing with $m_{2,l}$ steps takes place. On the lowest level l_{min}, in line 2, a nonlinear solution algorithm is applied to the nonlinear problem. In this work, the solver algorithm $BWE_l(\cdot,\cdot)$ on l_{min} is the same algorithm as the one used as a smoother in line 4 and 10.

Figure 4.2 shows both cycle types mentioned above. A four level V-cycle can be seen on the left. In this case, τ is equal to 1 and the top level l_{max} is defined as $l_{\text{min}} + 3$. The

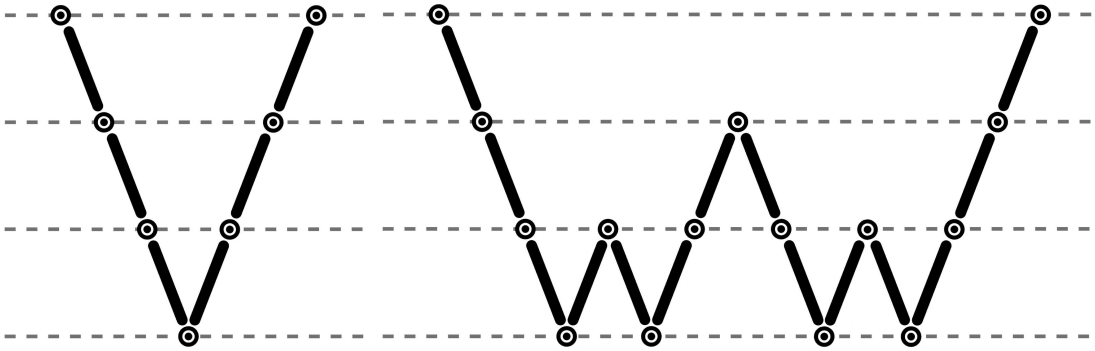

Figure 4.2: V-cycle with four levels (left), W-cycle with four levels (right).

dashed lines represent the levels on which the nonlinear multigrid algorithm is operating with the top dashed line representing the top level l_{max} and the bottom line representing the level l_{min}. The dots on the dashed lines represent an action rendered on this specific level, e.g. smoother steps or coarse level solving. In addition to that, the bold solid lines connecting the dots represent a transfer of information between the various levels, as seen in **Algorithm 1** line 5,6 and 9. Figure 4.2 also shows a four level W-cycle on the right. In this case, τ is equal to 2.

Algorithm 1 represents one multigrid cycle. These cycles are repeated until the norm of the nonlinear residual

$$\mathbf{R}_{l_{\text{max}}}(\mathbf{u}_{l_{\text{max}}}) := \mathbf{f}_{l_{\text{max}}} - \mathbf{L}_{l_{\text{max}}}(\mathbf{u}_{l_{\text{max}}}) \tag{4.2}$$

of the nonlinear problem $\mathbf{L}_{l_{\text{max}}}(\mathbf{u}_{l_{\text{max}}}) = \mathbf{f}_{l_{\text{max}}}$ is reduced to a desired value. Here, 10^{-12} is chosen as sufficient for all computations.

4.2 Nonlinear smoothers / single level solver

The choice of the smoother has one of the biggest impacts on the performance of a multigrid algorithm. There are many possibilities to choose a smoother and coarse level solver, but in our experience it has proven to be necessary to choose a rather efficient smoothing algorithm for turbulent flows and the associated meshes. The smoother is chosen in such a way that the nonlinear multigrid algorithm will accelerate the solution process in the early stages of the computation and then turn into Newton [54] like convergence rates at the end of the solution process.

4.2.1 Backward–Euler

In the following, the Backward–Euler pseudo-time stepping method will be considered, which can be characterized as a Rosenbrock method [60]. As soon as the pseudo-time step tends to infinity, the Newton algorithm [54] is recovered. As a fully implicit method, it requires the full Jacobian of the residual. The solver for the resulting linear system is typically chosen from the class of preconditioned iterative Krylov subspace methods. Due to the non-symmetric Jacobian (flexible) GMRes is a common choice [63], see Section 4.2.2. Since the DG approach achieves high order of accuracy via the introduction of additional degrees of freedom (DoF), it is not possible to use a lower order discretization to determine an approximation to the Jacobian like in finite volume methods as this would imply treating DoF corresponding to higher order modes explicitly. The Backward–Euler method is also the standard single level solver of the DLR PADGE-code [27] to solve the equations stated in (3.10) and is used as a reference.

Algorithm 2 Backward–Euler $BWE_l(\mathbf{u}_l, n, s)$

1: $i := 1$
2: $\mathbf{u}_{l,0} := \mathbf{u}_l$ /* set initial solution */
3: **while** $s \leq \|\mathbf{R}_l(\mathbf{u}_{l,i})\|$ **and** $i \leq n$ **do**
4: $\quad \Delta\mathbf{u}_{l,i} := \mathbf{u}_{l,i} - \mathbf{u}_{l,i-1}$
5: $\quad$ **Solve** $[(\mathsf{CFL}_i \Delta t_0)^{-1}\underline{\mathsf{M}} + \underline{\mathsf{J}}_l]\,(\Delta\mathbf{u}_{l,i}) = \mathbf{R}_l(\mathbf{u}_{l,i-1})$ /* solve linear problem */
6: $\quad \mathbf{u}_{l,i} := \mathbf{u}_{l,i-1} + \Delta\mathbf{u}_{l,i}$
7: $\quad i := i + 1$
8: **return** $\mathbf{u}_{l,i-1}$

Here $\underline{\mathsf{J}}_l$ is the fully implicit Jacobian matrix and $\underline{\mathsf{M}}$ is the mass matrix. In addition to that, $\mathbf{u}_{l,i}$ is a given state vector, with $\mathbf{u}_{l,i} \in \mathbf{V}_l \;\forall\; i \in \underline{n}$.

This method is controled by a pseudo-time stepping scheme $(\mathsf{CFL}_i \Delta t_0)$. Here Δt_0 is a local time steps computed from the local state and mesh size evaluated with a CFL number equal to one. The CFL_i scales this time step according to iteration i with the formular given in (4.3). The pseudo-time step acts as a stabilizing mechanism and for $\mathsf{CFL}_i \to \infty$ the Newton algorithm is recovered. Smaller time steps and accordingly smaller CFL numbers are required in the initial phase of the solution process when the current iterate solution approximation is still too far from the converged solution. A so called *switched evolution relaxation* (SER) [50] technique is employed to modify the CFL number during the solution process, enabling the algorithm to recover the optimal behavior of the Newton algorithm in the final phase of the solution process,

$$\mathsf{CFL}_i = \max\left(\mathsf{CFL}_0, \mathsf{CFL}_0 \left(\frac{\|\tilde{\mathbf{R}}_l(\mathbf{u}_{l,0})\|_{\mathsf{max}}}{\|\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})\|_{\mathsf{max}}}\right)^{\beta}\right), \tag{4.3}$$

where CFL_0 is the initial and minimal CLF number and β an exponent to control the growth rate of the CLF number. Since CFL_i increases as the residual norm $\|\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})\|_{\max}$ decreases over the iterations i, the mass matrix contribution in **Algorithm 2** vanishes as the nonlinear residual norm approaches zero. Here every entry j of $\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})$ corresponds to the discrete l^2-norm of the residual vector $\mathbf{R}_l(\mathbf{u}_{l,i})$ over the whole mesh for component j of the system of equations. Hence, the size of $\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})$ is $d+4$ for the RANS-$k\omega$ equations and $d+3$ for the RANS-SA equations. Each entry in $\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})$ is a vector l^2-norm over the whole mesh of the residual of one conservative variable at iteration i. Moreover, all entries of $\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})$ are normalized with the associated freestream conservative variable residual entries. Furthermore, $\|\cdot\|_{\max}$ denotes the maximum norm over the entries in $\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})$. For efficiency reasons the entries of $\tilde{\mathbf{R}}_l(\mathbf{u}_{l,i})$ for the k-equation in the RANS-$k\omega$ computations and for the $\tilde{\nu}$-equation in the RANS-SA computations are not considered in $\|\cdot\|_{\max}$. This is due to the fact, that these residual entries are normally very small initially and tend to start to rise at the beginning, thus preventing the CFL number to grow at the initial phase of the computation.

This Backward–Euler **Algorithm 2** is the nonlinear smoother and the solver for the coarsest level $l_{\min}$ within the proposed nonlinear multigrid **Algorithm 1**. Moreover, it is used as a benchmark of a single grid solver in Chapter 5.

Matrix-free implementation

While it usually pays off to assemble and store the full Jacobian with respect to the required overall run time of the solver, this leads to a considerable memory requirement. Fortunately, the GMRes algorithm requires only matrix–vector products, which can be implemented in a matrix-free fashion using finite differences of the residual. Either a central approximation,

$$\left[(\mathrm{CFL}_i\Delta t_0)^{-1}\underline{\mathsf{M}} + \underline{\mathsf{J}}_l\right]\mathbf{x} \approx \frac{\mathbf{L}_l(\mathbf{u}_l+\varepsilon\mathbf{x}) - \mathbf{L}_l(\mathbf{u}_l-\varepsilon\mathbf{x})}{2\varepsilon} + (\mathrm{CFL}_i\Delta t_0)^{-1}\underline{\mathsf{M}}\mathbf{x}, \tag{4.4}$$

which is second order accurate, or a simple one-sided first order accurate approximation,

$$\left[(\mathrm{CFL}_i\Delta t_0)^{-1}\underline{\mathsf{M}} + \underline{\mathsf{J}}_l\right]\mathbf{x} \approx \frac{\mathbf{L}_l(\mathbf{u}_l+\varepsilon\mathbf{x}) - \mathbf{L}_l(\mathbf{u}_l)}{\varepsilon} + (\mathrm{CFL}_i\Delta t_0)^{-1}\underline{\mathsf{M}}\mathbf{x}, \tag{4.5}$$

can be used. While the advantage of (4.5) is that $\mathbf{L}_l(\mathbf{u}_l)$ is constant during one linear solution process and thus does not have to be recomputed. Several, in particular turbulent, flow cases have been observed in which the additional accuracy of (4.4) was required for stability reasons. The parameter ε in the difference quotient has to be chosen sufficiently large to avoid cancellation errors but also sufficiently small to provide an accurate approximation of the derivative. Considering the relative scales of $\mathbf{u}_l$ and $\mathbf{x}$, a reasonable choice is given by $\varepsilon = \varepsilon_0\frac{\|\mathbf{u}_l\|}{\|\mathbf{x}\|}$, with a constant ε_0 that depends on the floating point accuracy, $\varepsilon_0 = 10^{-6}$ in here.

With this in mind, **Algorithm 2** can be implemented, depending on the preconditioner for the GMRes iteration, completely matrix free. Hence, memory consumption can be traded in for run time because of the additional residual computations for the matrix free formulation. Please note that this fact can only be utilized if the preconditioner for the GMRes iteration can be implemented memory efficient as well. This fact will be addressed in more detail in Section 4.4.

4.2.2 GMRes method

This section addresses the properties of the Krylov subspace method employed, namely the GMRes algorithm [63]. In Section 4.2.1, the GMRes algorithm is mentioned as a solver for the linear problems resulting from the Backward–Euler **Algorithm 2**. The following properties of the GMRes method are important for the numerical investigations in Chapter 5.

Let the linear problem be defined as $\mathcal{L}_l \mathfrak{u}_l = \mathfrak{f}_l$, with $\mathcal{L}_l \in \mathbb{R}^{n \times n}$ and $\mathfrak{f}_l, \mathfrak{u}_l \in \mathbb{R}^{n \times 1}$. Since the GMRes method is a Krylov subspace method, an approximate solution of the linear problem can be written as $\mathcal{L}_l^{-1} \mathfrak{f}_l \approx \mathfrak{u}_{l,i-1} = \mathfrak{u}_{l,0} + p_{i-1}(\mathcal{L}_l)\tilde{r}_0$, where $\tilde{r}_0 = \frac{r_0}{||r_0||_2}$ is the linear residual $r_0 = \mathfrak{f}_l - \mathcal{L}_l \mathfrak{u}_{l,0}$ divided by its norm and $p_{i-1}(x)$ is a polynomial of degree $i-1$.

The approximate solutions $\mathfrak{u}_{l,i}$ are vectors in the Krylov subspaces

$$\mathcal{K}_i(\mathcal{L}_l, \tilde{r}_0) := \text{span}\{\tilde{r}_0, \mathcal{L}_l \tilde{r}_0, \mathcal{L}_l^2 \tilde{r}_0, ..., \mathcal{L}_l^{i-1} \tilde{r}_0\}.$$

Moreover, a minimization problem in the Krylov subspace is solved reducing the Euclidian norm of the residual $\mathfrak{f}_l - \mathcal{L}_l \mathfrak{u}_{l,i}$ in a monoton fashion per iteration i. Hence, if the span of the Krylov subspace is $\mathbb{R}^n$, the exact solution of the linear problem can be recovered. For the GMRes method, optimality conditions are derived for the norm of the residual in the Krylov subspace. Moreover, the convergence of the GMRes method can be accelerated if the eigenvalues of the matrix $\mathcal{L}_l$ cluster away from zero. This leads to the common combination of the GMRes method with a preconditioner, since a preconditioner can alter the Eigendecomposition of $\mathcal{L}_l$. Please note that the GMRes method does not take into account lower level approximations unless this issue is addressed by the preconditioner used. This method can be seen as a black box approach to solve a linear system of equations. The chosen preconditioners are addressed in the next sections. As an alternative, the GMRes method can be implemented with a restart function which will set up a new Krylov subspace around the iterated residual $\tilde{r}_i$. Such a restart is performed in order to save memory, because more values need to be stored as the size of the Krylov subspace increases. However, in this thesis a GMRes method without restart functionality is employed.

4.3 Linear multigrid algorithm

In this work, a linear multigrid approach will be considered in order to improve the solution process of the linear problems resulting from **Algorithm 2**. A linear multigrid algorithm could be used instead of a GMRes scheme to solve the linear equations. Furthermore, a linear multigrid algorithm can be used as a preconditioner for the GMRes method.

Let $\mathcal{L}_l \mathfrak{u}_l = \mathfrak{f}_l$ denote a linear problem on level l.

Algorithm 3 Linear Multigrid Algorithm $LMG(\mathfrak{f}_l, \mathfrak{u}_l, \mathbf{m}_1, \mathbf{m}_2, l, l_{\text{min}}, \tau)$

```
 1: if l = l_min then
 2:     ũ_{l_min} = LJ_{l_min}(u_{l_min}, m_{2,l_min})
 3: if l > l_min then
 4:     u_l := LJ_l(u_l, m_{1,l})      /* pre-smoothing */
 5:     u^0_{l-1} := 0
 6:     f_{l-1} := I_l^{l-1} (f_l − L_l u_l)
 7:     for k = 1 to τ do
 8:         u^k_{l-1} := LMG(f_{l-1}, u^{k-1}_{l-1}, m_1, m_2, l − 1, l_min, τ)
 9:     ũ_l := u_l + I^l_{l-1} (u^τ_{l-1})
10:     ũ_l := LJ_l(u_l, m_{2,l})      /* post-smoothing */
11: return ũ_l
```

The linear multigrid algorithm is defined recursively in the same way as the nonlinear multigrid **Algorithm 1**. As usual, l denotes the current level of the linear multigrid algorithm and l_{min} the minimum level. Also $\mathbf{m}_1$ and $\mathbf{m}_2$ are represented by arrays which hold the number of smoother steps separately for each level, as seen for the nonlinear multigrid algorithm. For $l > l_{\text{min}}$, the linear state vector $\mathfrak{u}_l$ is smoothed $m_{1,l}$ times by a smoother algorithm $LJ_l(\cdot,\cdot)$, see line 4 of **Algorithm 3**. In contrast to the nonlinear multigrid algorithm, the state vector of level $l-1$ is set to zero and the restricted right hand side $\mathfrak{f}_{l-1}$ is set to the restricted residual of level l, see line 5-6. This is the main difference between the linear and nonlinear multigrid algorithms. The linear multigrid **Algorithm 3** can be derived from the nonlinear multigrid **Algorithm 1** by setting $\mathbf{u}^0_{l-1} := 0$ instead of $\mathbf{u}^0_{l-1} := \hat{I}^{l-1}_l \mathbf{u}_l$ for the nonlinear multigrid algorithm, since the lines 6 & 9 from **Algorithm 1** are then equal to the lines 6 & 9 in **Algorithm 3**.

The recursion in the lines 7-8 is defined the same way as for the nonlinear multigrid **Algorithm 1** and produces the same cycle types as seen in Figure 4.2. After the recursion, the prolongated state vector from level $l-1$ is added to the state vector on level l in line 9. In line 10, $m_{2,l}$ post smoothing steps are applied. On the level l_{min}, the same smoother algorithm $LJ_l(\cdot,\cdot)$ is used also as linear solver, see line 2.

An additional cycle type, as shown in Figure 4.3, will be investigated later in this work. The so called sawtooth-cycle is defined as a V-Cycle where all entrys in m_1 are equal to zero.

This means that no pre-smoothing steps are performed in the linear multigrid algorithm cycle.

In the case of applying a GMRes method to the linear problems, either the linear multigrid **Algorithm 3** or the linear smoother $LJ_l(\cdot,\cdot)$ is used as a preconditioner. Therefore, the only difference between the two investigated preconditioners is that the linear multigrid additionally uses discretizations from lower levels. Moreover, if a single level preconditioner $LJ_l(\cdot,\cdot)$ is applied it is the same algorithm as if a linear multigrid with only one level is employed. **Algorithm 3** represents one linear multigrid cycle. These cycles are repeated until the norm of the linear residual $r_l = \mathfrak{f}_l - \mathcal{L}_l \mathfrak{u}_l$ on level l is reduced under a certain threshold or until a given number of iterations is performed. Investigations concerning the linear multigrid algorithm are shown in Chapter 5.

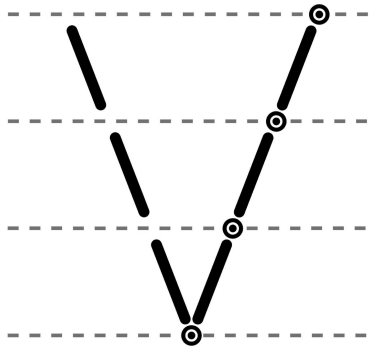

Figure 4.3: Sawtooth-cycle with four levels.

4.4 Linear smoothers / preconditioner

In this section a smoother for the linear multigrid **Algorithm 3** is introduced, which can also be applied as a single level preconditioner for the GMRes iterations in the Backward–Euler **Algorithm 2**. The storage of the system matrix can be avoided with the matrix-free implementation technique of **Algorithm 2** and the GMRes method. However, a good preconditioner is still required to allow for reasonable convergence rates for the linear problems in the Krylov process. For matrix-based implementations, typically incomplete LU factorizations provide a good compromise between memory requirement and performance. In particular, the version with no additional fill-in, ILU(0), is often proposed [14, 27]. It requires the same storage as the original system matrix. For a matrix-free implementation, this preconditioner is the memory bottleneck. Moreover, in case of a domain-decompositioned parallelization, the ILU preconditioner becomes weaker because it can then only be applied on each subdomain. Hence, this preconditioner may not be sufficiently strong for solving the stiff problems associated with turbulent flows. These drawbacks can be tackled by the linear solver $LJ_l(\cdot,\cdot)$ proposed in this section, which is later applied as a preconditioner or smoother.

A semi-implicit method is considered which keeps the important couplings in the system matrix while dropping others. In particular, lines of elements are chosen, i. e. disjoint subsets of elements. The approach, how to compute lines, and how to approximate the

Jacobian including the important couplings with the help of lines will be explained in Section 4.7. The inversion on a line-based system matrix can be applied exactly, e. g. with the Thomas algorithm [69], whereas ILU represents an approximate inverse only. However, the underlying matrix to be inverted is only an approximation of the fully implicit matrix. Thus, in both cases the inverse of the Jacobian is in-exact w. r. t. the original matrix and it is not a priori clear which approach yields better results. However, the semi-implicit method will keep its properties under a domain-decompositioned parallelization because the domain-decomposition can be chosen in such a way that the lines of elements are not cut by a domain border.

A Jacobi-type iteration on the full matrix is used, where the inverted part of the matrix is chosen as the subset of the full matrix contained in lines. Let $\mathcal{L}_l = (\mathsf{CLF}_i \Delta t_0)^{-1} \underline{\mathsf{M}} + \underline{\mathsf{J}}_l$ denote the full system matrix, $\underline{\mathsf{L}}_l$ the sub-matrix extracted along lines, and $\underline{\mathsf{N}} = \mathcal{L}_l - \underline{\mathsf{L}}_l$ the remaining matrix blocks, then an iterative Jacobi-type scheme for the linear system $\mathcal{L}_l \mathbf{x} = \mathfrak{f}_l$ is given by

$$\mathbf{x}^{(k+1)} = \underline{\mathsf{L}}_l^{-1} \left[\mathfrak{f}_l - \underline{\mathsf{N}} \mathbf{x}^{(k)} \right]. \tag{4.6}$$

Subtracting the last iterate $\mathbf{x}^{(k)}$ on both sides of (4.6) and introducing $\Delta \mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$, the process can be written in terms of $\underline{\mathsf{L}}_l$ instead of $\underline{\mathsf{N}}$, eliminating the need to form the latter explicitly:

$$\Delta \mathbf{x}^{(k+1)} = \underline{\mathsf{L}}_l^{-1} \left[\mathfrak{f}_l - \mathcal{L}_l \mathbf{x}^{(k)} \right], \qquad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta \mathbf{x}^{(k+1)}. \tag{4.7}$$

Again, it is not necessary to store and compute the whole Jacobian, as the required matrix-vector product can be evaluated in a matrix-free fashion via finite differences, as seen for the Backward–Euler **Algorithm 2**. Using this approach intra-line couplings are respected exactly via the inversion along lines, whereas inter-line couplings are treated in a weaker form via multiplication with the full system matrix to form the new linear residual. Typically, only very few iterations are taken. The outer Krylov iteration takes care of all error modes that cannot efficiently be reduced using the line-Jacobi iteration.

Let $\mathcal{L}_{l,k}(\mathfrak{u}_{l,k}) = \mathfrak{f}_{l,k}$ be the underlying linear problem on line k, then the Line–Jacobi method can be stated as follows.

Algorithm 4 Line–Jacobi method $LJ_l(\mathfrak{u}_{l,k,0}, n)$

1: **for** $i = 1$ **to** n **do**
2: $\quad \Delta \mathfrak{u}_{l,k,i} := \mathfrak{u}_{l,k,i} - \mathfrak{u}_{l,k,i-1} = \underline{\mathsf{L}}_{l,k}^{-1} (\mathfrak{f}_{l,k} - \mathcal{L}_{l,k} \mathfrak{u}_{l,k,i-1})$
3: $\quad \mathfrak{u}_{l,k,i} := \mathfrak{u}_{l,k,i-1} + \Delta \mathfrak{u}_{l,k,i}$
4: **return** $\mathfrak{u}_{l,k,n}$

Here $\underline{\mathsf{L}}_{l,k}^{-1}$ is the inverse of the system matrix computed one line k in the mesh.

Matrix-free implementation

The Jacobian on one element lines consist of a single block entry per element, whereas an inner element of a longer line contributes three blocks: one for inter-element couplings and one additional block for both the preceding and subsequent element in the same line. Thus, depending on the number and length of lines, the memory consumption for storing the Jacobian terms of the line-based matrix-free solver is between one and three blocks per element in the mesh. In contrast to that, the matrix-based full Jacobian solver requires $2(1 + N_{\text{nb}})$ blocks per element, neglecting mesh boundaries. Here, N_{nb} is the number of neighbors per element, $N_{\text{nb}} = 4$ for quadrilaterals and $N_{\text{nb}} = 6$ for hexahedra. The factor of 2 arises from the need to store both the original matrix and the ILU(0) preconditioner. Thus, memory savings between 70 % and 90 % can be expected for structured meshes in 2D. In 3D, the savings increase to values between 79 % and 93 %. Note, however that every saving of memory through a matrix-free implementation of all proposed algorithms always comes with the price of a longer overall CPU time in comparison to the matrix-based versions of these algorithms. A matrix-free single level Backward–Euler using a GMRes method preconditioned with a Line–Jacobi method runs typically 2 times slower than the matrix-based version in this framework.

Algorithm 4 can be used as a preconditioner within the GMRes method and also as a smoother in the linear multigrid algorithm.

4.5 Startup strategy

Fine mesh computations at a high polynomial degree are known to be challenging in DG methods. Even for implicit methods this is true due to the stiffness of the discrete equations and the lack of an appropriate initial solution. Thus, such computations may be computing time intensive to converge to a satisfying state. This effect will be demonstrated in Chapter 5.

An appropriate startup strategy addresses this issue by providing an appropriate initial solution for the top level of the solution process. The basic idea is to start the computation on the lowest level and transfer the resulting solution to the next higher level to provide a suitable initial solution in comparison to the freestream state vector. For DG codes order sequencing as a startup strategy is the most common approach and will here be referred to as p-startup strategy. If a nested hierarchy of meshes can be provided, mesh sequencing as a startup strategy can be addressed even with a none agglomeration DG code. However, in the proposed framework both strategies are feasible for every given mesh. An h-startup strategy can be realized via agglomeration of the given fine mesh and even p-startup strategies on mesh agglomerates are possible.

Algorithm 5 describes the startup algorithm implemented within this framework. Let $\mathbf{L_l}(\mathbf{u_l}) = \mathbf{f_l}$ be the nonlinear problem on level l with the nonlinear residual $\mathbf{R}_l(\mathbf{u}_l)$.

Algorithm 5 Startup strategy $Startup(l_{\text{min}}, l_{\text{max}}, r_{\text{tol}}, b_{\text{sg}}, b_{\text{it}})$

```
1:  u_{l_min,0} = u_{l_min}
2:  j := 0
3:  k := l_min
4:  b_1 := true
5:  for i = l_min to l_max − 1 do
6:      if b_sg == true then
7:          k := i
8:      while r_tol < ||R_i(u_{i,j})|| and b_1 == true do
9:          j := j + 1
10:         u_{i,j} := NMG(f_i, u_{i,j−1}, m_1, m_2, i, k, τ)
11:         if b_it == false then
12:             b_1 := false
13:     u_{i+1,0} := Î_i^{i+1} u_{i,j}
14:     j := 0
15:     b_1 := true
16: return u_{l_max,0}
```

In order to save computational time, a residual tolerance r_{tol} can be provided for every level of the startup strategy, see line four of **Algorithm 5**. With the help of this residual tolerance, it will be investigated how much a solution has to be converged on the lower levels in order to be a valid initial solution for the next level of the startup strategy. An analysis of how much the residual on every sublevel of the startup strategy has to be reduced to be effective in terms of CPU time can be found in Section 5.2.2.

Moreover, there is an option to perform one nonlinear multigrid iteration only on each level of the startup strategy by setting b_{it} to false. This option will also be applied to problems in Chapter 5, mostly when the Reynolds-Averaged Navier-Stokes (RANS)-SA equations are solved.

The nonlinear multigrid algorithm can be applied as a single grid solver in the startup strategy by setting b_{sg} to true. Otherwise, the nonlinear multigrid algorithm will always utilize all available lower levels for $l_{\text{min}} \leq i$, which is referred to as a full multigrid algorithm (FMG) [70].

In the given implementation, the **Algorithm 5** can produce converged solutions on every sublevel, e. g. for a p-startup strategy it can generate solutions on the same computational mesh for all discretization from p_{min} to p_{max}. For an h-startup strategy, the initial fine mesh will be coarsened in a way that the resulting coarse meshes will build a nested hierarchy of meshes. **Algorithm 5** can then compute a solution on the coarsest mesh and use this as an initial solution for the next finer mesh in the hierarchy until the algorithm produces

a solution for the initial fine mesh. Furthermore, a p-startup strategy is applied on the coarsest mesh of every h-startup strategy in the given framework.

4.6 Transfer operators

One of the main differences between a p- or h-multigrid algorithm is the transfer operator. In a p-multigrid algorithm data is transferred between discretizations with different polynomial degrees on the same mesh, whereas in a h-multigrid algorithm data is transferred between different meshes for the same polynomial degree. In the following, both transfer methods will be introduced.

The p-multigrid method requires transfer operators between different linear spaces $\mathbf{V}_l$, where $\mathbf{V}_l$ are the solution spaces of the DG discretization on a given mesh and with a given polynomial degree p for level l. On a fixed computational mesh, it holds that $\mathbf{V}_{l-1} \subset \mathbf{V}_l$, where $\mathbf{V}_{l-1}$ is a DG discretization on the same mesh with polynomial degree $p-1$. Therefore, only p-transfer operations for discretizations on the same computational mesh are considered. The solution and the defect between the various approximation levels have to be transferred for a nonlinear multigrid algorithm. The restriction operator for both, h- and p-multigrid algorithms of the nonlinear state vector, is given by an orthogonal L^2-projection $\hat{I}_l^{l-1} : \mathbf{V}_l \to \mathbf{V}_{l-1}$ [7]. Furthermore, the prolongation of the nonlinear state vector is obtained via a natural injection $I_{l-1}^l : \mathbf{V}_{l-1} \to \mathbf{V}_l$ and a restriction operator for the defect between the various approximation levels is defined by $I_l^{l-1} := (I_{l-1}^l)^\top$.

The DG transfer operators are particularly simple if ortho-normal hierarchic basis functions are used on each level. Then, for instance, for a p-multigrid algorithm, the prolongation operator can have the form of an identity matrix filled with additional zero rows in the bottom, depending of the current implementation in the code.

When the Jacobian is needed on lower levels in a multigrid algorithm with a Backward–Euler solver as smoother the Galerkin transfer [70] can be considered. Whit in the Galerkin transfer no additional transfer operators are required to transfer the Jacobian matrix from one level to another. It is defined as $\underline{\mathsf{J}}_{l-1} = I_l^{l-1} \underline{\mathsf{J}}_l I_{l-1}^l$ for a given matrix $\underline{\mathsf{J}}_l$ consisting of vectors which are in $\mathbf{V}_l$. Then the resulting matrix $\underline{\mathsf{J}}_{l-1}$ consists of vectors which are in $\mathbf{V}_{l-1}$.

The h-transfer operators for an h-multigrid will only be introduced on a nested hierarchy of meshes, generated by agglomeration of the finest mesh. The agglomeration is formulated in an unstructured way making no use of potentially embedded coarse structured meshes. Thus, coarse levels of structured meshes are not structured in general. In all cases, the coarser meshes consist of agglomerates which in turn are subsets of the cells

of the underlying fine mesh. For this reason, the statement $\mathbf{V}_{l-1} \subset \mathbf{V}_l$ holds also for the h-multigrid case if the polynomial degree in both spaces is the same.

The transfer operators can be derived for p- and h-transfers in one general formulation as follows.

Definition 4.1:

Like in Chapters 2 & 3, let Ω be a physical space, open, bounded, and connected.

Then Ω can be subdivided into s_{l-1} open, disjoint subdomains $\kappa_{l-1,m}$ with $m \in \underline{s}_{l-1}$. The set of all $\kappa_{l-1,m}$ is called $\mathcal{T}_{l-1}$, with

$$\overline{\Omega} = \bigcup_{\kappa_{l-1,m} \in \mathcal{T}_{l-1}} \overline{\kappa}_{l-1,m}\,, \quad \kappa_{l-1,n} \cap \kappa_{l-1,m} = \emptyset \quad , n, m \in \underline{s}_{l-1}\,,\, n \neq m.$$

Furthermore, let assume every $\kappa_{l-1,m} \in \mathcal{T}_{l-1}$ can be subdivided into $s_{l-1,m}$ open disjoint subdomains κ_{l,m_k}. Hence, the following holds

$$\overline{\kappa}_{l-1,m} = \bigcup_{k \in \underline{s}_{m,l-1}} \overline{\kappa}_{l,m_k}\,, \quad \kappa_{l,m_k} \cap \kappa_{l,m_g} = \emptyset \quad , k, g \in \underline{s}_{m,l-1}\,,\, k \neq g.$$

Please note that each domain $\kappa_{l-1,m}$ represents an agglomerate of mesh elements κ_{l,m_k}. Hence, in the p-multigrid case $s_{l-1,m}$ will always be equal to one because the computational mesh will stay the same.

Let $\{\mathbf{v}_l^j\}$ be a basis of $\mathbf{V}_l$ with $\dim(\mathbf{V}_l) = n_l$ and $\mathbf{V}_l$ is defined on $\mathcal{T}_l$ with

$$\overline{\mathcal{T}_l} = \bigcup_{m \in \underline{s}_{l-1}} \bigcup_{k \in \underline{s}_{m,l-1}} \overline{\kappa}_{l,m_k} = \overline{\Omega}.$$

Analogously, let $\{\mathbf{v}_{l-1}^j\}$ be a basis of $\mathbf{V}_{l-1}$ with $\dim(\mathbf{V}_{l-1}) = n_{l-1}$ and $\mathbf{V}_{l-1}$ is defined on $\mathcal{T}_{l-1}$ with

$$\overline{\mathcal{T}_l} = \bigcup_{m \in \underline{s}_{l-1}} \overline{\kappa}_{l-1,m} = \overline{\Omega}.$$

Then, $\mathbf{u}_l \in \mathbf{V}_l$ can be written as $\mathbf{u}_l = \sum_{j=1}^{n_l} u_l^j \mathbf{v}_l^j$, with $u_l^j \in \mathbb{R}$ and $\mathbf{u}_{l-1} \in \mathbf{V}_{l-1}$ can be written as $\mathbf{u}_{l-1} = \sum_{j=1}^{n_{l-1}} u_{l-1}^j \mathbf{v}_{l-1}^j$, with $u_{l-1}^j \in \mathbb{R}$.

With this in mind the orthogonal L^2-projection $\hat{I}_l^{l-1} : \mathbf{V}_l \to \mathbf{V}_{l-1}$ for an h- and p-transfer operator can be derived as follows:

$$
\begin{aligned}
& \int_\Omega \mathbf{v}_{l-1}^i(\mathbf{u}_l - \mathbf{u}_{l-1})\,\mathrm{d}V && = 0 && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Leftrightarrow\ & \sum_{m=1}^{s_{l-1}} \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i(\mathbf{u}_l - \mathbf{u}_{l-1})\,\mathrm{d}V && = 0 && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Leftrightarrow\ & \sum_{m=1}^{s_{l-1}} \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i \mathbf{u}_l\,\mathrm{d}V && = \sum_{m=1}^{s_{l-1}} \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i \mathbf{u}_{l-1}\,\mathrm{d}V && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Leftrightarrow\ & \sum_{m=1}^{s_{l-1}} \sum_{k=1}^{s_{l-1,m}} \int_{\kappa_{l,m_k}} \mathbf{v}_{l-1}^i \mathbf{u}_l\,\mathrm{d}V && = \sum_{m=1}^{s_{l-1}} \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i \mathbf{u}_{l-1}\,\mathrm{d}V && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Leftrightarrow\ & \sum_{m=1}^{s_{l-1}} \sum_{k=1}^{s_{l-1,m}} \int_{\kappa_{l,m_k}} \mathbf{v}_{l-1}^i \sum_{j=1}^{n_l} u_l^j \mathbf{v}_l^j\,\mathrm{d}V && = \sum_{m=1}^{s_{l-1}} \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i \sum_{j=1}^{n_{l-1}} u_{l-1}^j \mathbf{v}_{l-1}^j\,\mathrm{d}V && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Leftrightarrow\ & \sum_{m=1}^{s_{l-1}} \sum_{k=1}^{s_{l-1,m}} \sum_{j=1}^{n_l} u_l^j \int_{\kappa_{l,m_k}} \mathbf{v}_{l-1}^i \mathbf{v}_l^j\,\mathrm{d}V && = \sum_{m=1}^{s_{l-1}} \sum_{j=1}^{n_{l-1}} u_{l-1}^j \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i \mathbf{v}_{l-1}^j\,\mathrm{d}V && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Leftrightarrow\ & \sum_{j=1}^{n_l} u_l^j \sum_{m=1}^{s_{l-1}} \sum_{k=1}^{s_{l-1,m}} \int_{\kappa_{l,m_k}} \mathbf{v}_{l-1}^i \mathbf{v}_l^j\,\mathrm{d}V && = \sum_{j=1}^{n_{l-1}} u_{l-1}^j \sum_{m=1}^{s_{l-1}} \int_{\kappa_{l-1,m}} \mathbf{v}_{l-1}^i \mathbf{v}_{l-1}^j\,\mathrm{d}V && \forall\, \mathbf{v}_{l-1}^i \in \mathbf{V}_{l-1} \\
\Rightarrow\ & \mathbf{M}^{l-1,l}\mathbf{w}_l && = \mathbf{M}^{l-1,l-1}\mathbf{w}_{l-1} \\
\Leftrightarrow\ & \underbrace{(\mathbf{M}^{l-1,l-1})^{-1}\mathbf{M}^{l-1,l}}_{:=\hat{I}_l^{l-1}}\mathbf{w}_l && = \mathbf{w}_{l-1},
\end{aligned}
$$

where $\mathbf{M}^{l-1,l-1}, \mathbf{M}^{l-1,l}$ are scalar product matrices and $\mathbf{w}_l$ is a vector consisting of all $u_l^j \in \mathbb{R}$ from $\mathbf{u}_l = \sum_{j=1}^{n_l} u_l^j \mathbf{v}_l^j$. The same can be done for the natural injection $I_{l-1}^l : \mathbf{V}_{l-1} \to \mathbf{V}_l$. This will lead to

$$
\begin{aligned}
\Rightarrow\ & \mathbf{M}^{l,l-1}\mathbf{w}_{l-1} && = \mathbf{M}^{l,l}\mathbf{w}_l \\
\Leftrightarrow\ & \underbrace{(\mathbf{M}^{l,l})^{-1}\mathbf{M}^{l,l-1}}_{:=I_{l-1}^l}\mathbf{w}_{l-1} && = \mathbf{w}_l.
\end{aligned}
$$

Therefore, it follows that $I_l^{l-1} = \mathbf{M}^{l-1,l}(\mathbf{M}^{l,l})^{-tr}$ is a restriction operator for the defect between the various approximation levels.

The basis functions $\{\mathbf{v}_l^i\}$ are orthonormal on every open domain κ_{l,m_k} and the same is true for the basis $\{\mathbf{v}_{l-1}^i\}$ on every $\kappa_{l-1,m}$, as stated in Section 3.4. Hence, the transfer operators I_l^{l-1} and $\hat{I}_l^{l-1}$ are identical because for this specific choice of basis functions the matrices $(\mathbf{M}^{l-1,l-1})^{-1}$ and $(\mathbf{M}^{l,l})^{-tr}$ are identity matrices.

4.7 Line creation

In this section several algorithms are discussed as how to create the lines which are used in the Line–Jacobi **Algorithm 4**. One way to create lines is discussed in this section. Recall that, with the help of lines consisting of mesh elements, it is possible to get a memory efficient approximation for the Jacobian which can be inverted exactly. Moreover, lines can be preserved under domain-decompositioned parallelization, which is beneficial for the Line–Jacobi **Algorithm 4** because the performance of this algorithm will not change

for different domain-decompositions. A line of elements in the computational mesh is defined as follows.

Definition 4.2:

A line of mesh elements fulfills the following properties:

- A line is contiguous.
- A line has at most one direct coupling between two line elements.
- A line has no closed loops.

Selecting the system matrix blocks for elements on the diagonal as well as interface blocks corresponding to couplings over a common interface between neighboring elements within a line yields a tri-diagonal block system per line, see Figure 4.5, which can be inverted very efficiently, e. g. with the Thomas algorithm [69]. In order to find suitable lines, a scalar advection-diffusion problem is defined and examined [23].

Consider, the advection-diffusion problem $\nabla \cdot (a \nabla \mathbf{u}) - b \cdot \nabla \mathbf{u} = 0$, on Ω. The weak formulation of this problem on a given mesh is given by:

Find $\mathbf{u} \in H^{m,b}(\mathcal{T}_h)$ such that

$$\begin{aligned} &\textstyle\int_{\Gamma_I \cap \Gamma} -a \, \nabla \mathbf{u} \cdot \mathbf{v} \, \mathrm{d}s + \int_{\mathcal{T}_h} a \, \nabla \mathbf{u} \cdot \nabla \mathbf{v} \, \mathrm{d}V \\ &\textstyle+ \int_{\Gamma_I \cap \Gamma} b \, \mathbf{u} \cdot \mathbf{v} \, \mathrm{d}s - \int_{\mathcal{T}_h} b \, \mathbf{u} \cdot \nabla \mathbf{v} \, \mathrm{d}V = 0 \quad \forall \mathbf{v} \in H^{m,b}(\mathcal{T}_h). \end{aligned} \tag{4.8}$$

where a is a chosen value, b is an initial or interim solution of the flow field to be computed.

The weak formulation 4.8 is discretized with a $p = 0$ DG method on the computational mesh with the same techniques as in Chapter 3, thus $\mathbf{u}_h, \mathbf{v}_h \in \mathcal{V}_h^0(\mathcal{T}_h)$. For a $p = 0$ DG discretization, only the two face integrals in 4.8 remain, since the derivative of a constant vanishes. The first integral of the DG discretization is approximated with a diffusive flux via the second scheme of Bassi and Rebay (BR2) [5, 10] and the second integral with an upwind flux between mesh elements.

Then the resulting system matrix of the DG discretization can be used as the weight matrix between the cells in the mesh, because due to ansatz and test functions being of a polynomial degree $p = 0$ every row in the matrix stands for one cell in the mesh. Every non-zero off-diagonal entry in a row can be interpreted as the edge-weight to a neighboring cell in the mesh. The two highest off-diagonal entries in every row indicate the neighboring cells with the strongest coupling to the current cell and the best candidates for creating a line. Because of the approximated diffusive flux via the BR2 scheme the lines which are created in the diffusion dominated part of the flow field align in the main

direction of diffusive effects, e. g. in boundary layers near solid walls the lines are orthogonal to those walls. Thus addressing mesh anisotropies, see Figure 4.4 (right). This is due to the fact that, the BR2 scheme applied to this discretization represents the inverse distance between mid points. In the convection dominated part, e. g. the far field, the lines follow the convective flow direction, see Figure 4.4 (left). This is due to the fact that b, which is used in the upwind flux of the DG discretization, is an initial or interim solution of the flow field to be computed. This ansatz tries to ensure that those couplings in the Jacobian which are most relevant for solving the DG discretization (3.10) are retained in the lines. On the other hand, weaker couplings are neglected in the line approximation of the Jacobian.

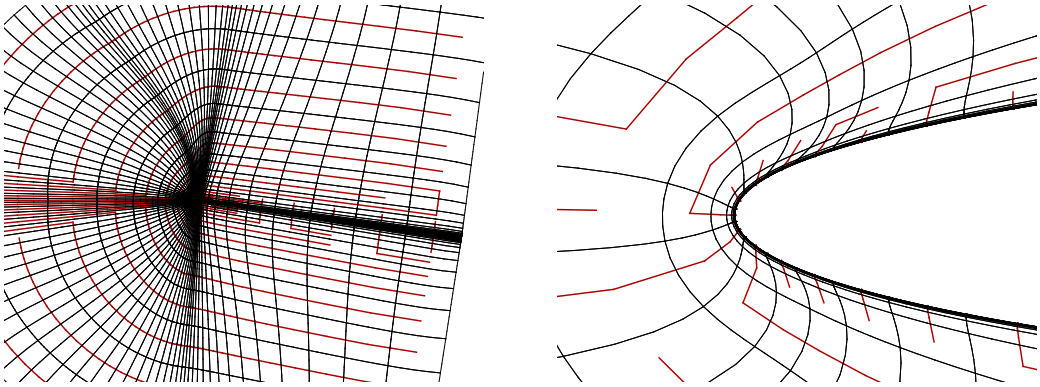

Figure 4.4: Computed lines (red) in a mesh of an airfoil, created with a scalar advection-diffusion problem

There exist other algorithms to find lines in the mesh, some of which follow a geometrical approach [38]. This approach uses the inverse distance between cell mid points, which will create lines addressing anisotropies in the mesh and is very similar to the effect of a scaled diffusion operator for a $p = 0$ DG discretization. Isotropic regions will be treated using lines which consist of only a single element. This geometrical approach also provides a weight matrix between the cells in the mesh. Again, every row can be associated with one cell in the mesh and every off-diagonal entry as a weight reference value to a neighboring cell. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{\mathbf{n}\times\mathbf{n}}$ the **Algorithm 6** is used to create the lines.

This algorithm can be implemented in a way that most of the time the resulting lines are independent of the cell numbering in the mesh. To achieve this the circles have to be cut on the smallest weight value. If the smallest weight value appears several times in the circle than the circle is cut on every occurrence of this value. The algorithm is not independent of the cell numbering if the second maximal entry in a row appears several times.

The third possibility is to use lines consisting of a single element only. This means that every cell in the mesh is a separate line. In this way, element- and line-implicit methods

Algorithm 6 Line Search Algorithm $LSA(\mathbf{W})$

for $i = 1$ to n **do**
    Find two max. off-diagonal entries $w_{i,j}$ and $w_{i,l}$ in row i of **W**.
    Save the colum indices j, l.
    Check if $w_{j,i}$ is one of two max. off-diagonal entries in row j of **W**.
    If true join cell i and j to a line.
    Check if $w_{l,i}$ is one of two max. off-diagonal entries in row l of **W**.
    If true join cell i and l to a line.
Check for circles in all lines and cut them.
return Lineset

are treated in a unified framework. However, the element-implicit approach it is not suited to address mesh anisotropies.

In general, lines defined via the advection-diffusion approach are solution-dependent and could be re-computed along with the original RANS problem for maximal effectivity. However, this complicates the code infrastructure, in particular if computations are to be done in parallel and lines should not cross domain boundaries. As a compromise, lines are only determined at the beginning of each computation on a new level, taking into account the flow field evaluated for a lower level discretization, e.g. from a startup strategy. The overall process is initiated with freestream velocity in the whole computational domain.

The matrix breakdown in Figure 4.5 shows all block entries of the full Jacobian, but only the blue entries are the Jacobian blocks which are considered after applying **Algorithm 6**. The diagonal darker blue entries would be used if all lines only contain one element, which would be a representation of an element-implicit method. The light blue off-diagonal blocks are contributions from neighbor elements of a line. As can be seen in Figure 4.5 (blue blocks), the Jacobian on lines is structured as follows: First all lines which contain more than one element only, see the dashed boxes in Figure 4.5, and then all lines which contain one element only as seen in the right lower corner of Figure 4.5. This chronology is introduced for implementational reasons and has no effect on the solution algorithm itself. Therefore, **Algorithm 6** creates an approximation of the full Jacobian by neglecting block entries. How this approximation is used can be seen in **Algorithm 4**.

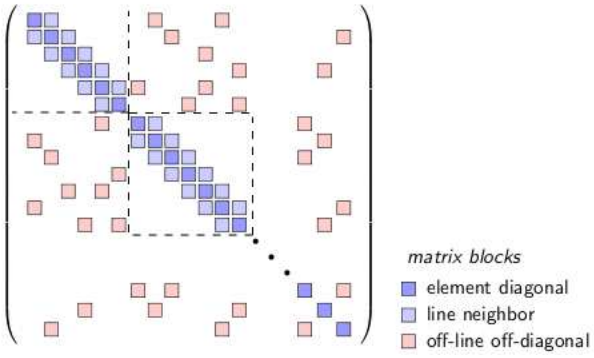

Figure 4.5: Jacobian breakdown for a quadrangle 2D mesh.

It is known in literature that a line-implicit approach is always superior to an element-implicit approach [14, 38, 40]. Moreover, experience shows that it is always beneficial to

use as many element couplings in a line as possible if they are computed via an advection-diffusion approach as stated in this section [76]. Therefore, beneficial effects while using lines are analyzed in Chapter 5.

4.8 Mesh agglomeration

For the h-version of the multigrid algorithm an agglomeration of the underlying fine mesh is performed [3, 4]. A coarse level is constructed by connecting a certain number of fine mesh elements to form a single agglomerate. No simplification of the resulting geometry is performed, thus ensuring the higher order boundary representation mentioned in Section 3.5. Furthermore, bounding faces between elements that are part of the same agglomerate can be ignored on coarse levels. Coarser levels are obtained by recursive application of the described procedure, i. e., the second agglomeration is performed based on the first level of agglomerates. Thus, a hierarchy of nested meshes is created, which is appropriate for the transfer operators described in Section 4.6. For the numerical examples, the agglomeration is based on the `MGridGen` software, which employs graph-based algorithms to optimize the ratio of surface area to the volume of the agglomerate [49], i. e., it aims at isotropic coarse meshes. For each mesh agglomeration procedure a coarsening rate c_{rt} can be provided which specifies how many mesh elements should be joined to create an agglomerated mesh element. An example for the agglomeration with coarsening rate $c_{\text{rt}} = 4$ can be seen in Figure 4.6. Here a mesh of the L1T2 three element airfoil test case is considered, which is introduced in detail in Section 5.1.

A theoretical evaluation of the cost scaling per operation of this framework on each agglomerate of a cartesian fine mesh can be performed as follows. The vectors and sparse matrices assembled over the whole mesh scale over the mesh size in a multigrid sense, since the coarsening rate c_{rt} contributes linearly to the size of these vectors and matrices. This reduces the cost of linear algebra operations on agglomerates. In particular, the size of the Jacobian matrix, which is assembled over the whole agglomerate and for which only non-zero entries are stored, reduces by about a factor of approximately $\sim c_{\text{rt}}$ in comparison to the agglomerate or mesh which is one level finer. For example the number of stored system matrix entries per equation for a $p = 2$ computation on the mesh and agglomerations shown in Figure 4.6 is 3 695 040, 1 328 832 and 415 836. Altough in Figure 4.6, a coarsening rate of $c_{\text{rt}} = 4$ is used the number of the non-zero matrix entries in this example is not reduced by a factor of $\sim \frac{1}{4}$ but by a factor of about $\sim \frac{1}{3}$. This is due to the fact that on a agglomerated coarse mesh the number of neighboring cells to a given cell is not restricted by 4 in contrary to the unstructured fine mesh, see Figure 4.6. Recall that, every neighboring element adds one block entry to the row representing an element in the system matrix. In summary, the cost of linear algebra operations scales on coarse

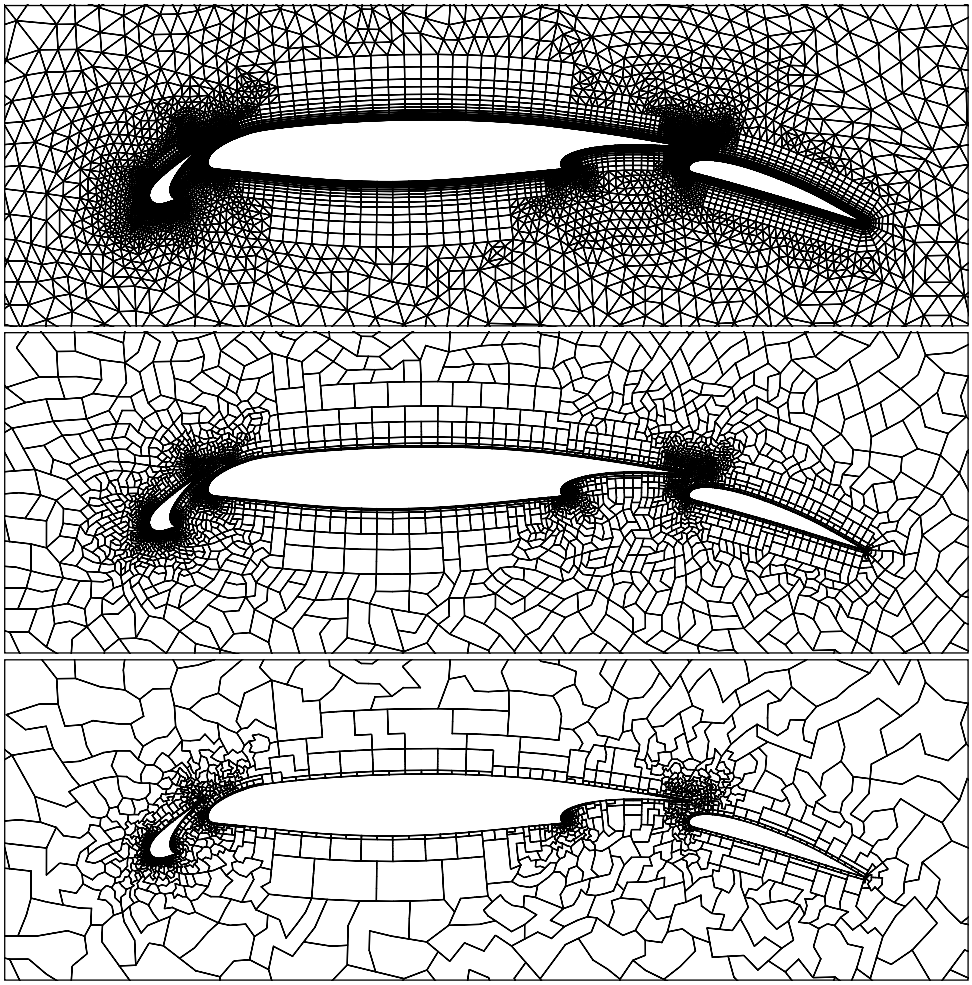

Figure 4.6: Unstructured L1T2 three element airfoil mesh with 23 824 elements and two agglomerations.

levels at most with the coarsening rate. This is in contrast to operator evaluations, e.g. integration on cells, which is considered next. The integration on cells of the agglomerated meshes is realized by integration over cells of the underlying fine mesh. During this integration, all quadrature points from the original fine mesh are considered. That means that the cost for this operation does not scale with the coarsening rate c_{rt}. However, due to the reduction of the number of faces to be considered, as fine elements are joined, the cost for face integration is reduced on coarser levels, depending on the agglomeration coarsening rate.

This gives some additional motivation for the use of particularly strong implicit solvers, which will spend a relatively large fraction of the overall time in linear algebra operations, see Table 4.1.

In Table 4.1 different settings for h-multigrid algorithms are considered. In every row, except the first one, one V-cycle of a nonlinear h-multigrid algorithm is performed. One smoother step with the Backward–Euler algorithm is applied on every level. As a linear solver on every level a linear h-multigrid preconditioned GMRes method is applied with

no.	levels	c_{rt}	CPU time			
			cycle	Jac.	GMRes	matrix mult.
1.	1	0	1.00	0.50	0.46	0.23
2.	2	2	1.73	0.49	1.09	0.56
3.	3	2	2.18	0.49	1.49	0.80
4.	3	4	1.31	0.50	0.65	0.35
5.	5	4	1.55	0.51	0.90	0.49
6.	5	2	2.56	0.51	1.81	1.00

Table 4.1: Nonlinear V-cycle iteration cost within the current framework: turbulent L1T2 three element airfoil test case on an unstructured mesh.

60 GMRes iterations, e.g. in row 2 of Table 4.1 over all levels 180 GMRes iterations are performed. The first row includes the computing time of two Backward–Euler iterations without the use of any multigrid algorithms and all time-dependent entries in Table 4.1 are normalized with the CPU "cycle" time of these two iterations. Two iterations are chosen for an easy comparison between the first row and all other rows in Table 4.1 because for a nonlinear V-cycle and one pre- and post-smoothing step on every level two Backward-Euler iterations are performed on the top level. Thereby, the amount of time that exceeds 1.00 in the column "cycle" of the rows 2 to 6 is the amount of time spent on the lower levels of the nonlinear multigrid algorithm. The sums of the computing times in the columns "Jac." and "GMRes" are always close to the time of the whole "cycle", since the assembling of the Jacobian and the linear solver are the two major parts of every nonlinear solution algorithm presented in this work. Since, most of the matrix-vector multiplications are performed in the linear solver. The entry in "matrix multi." is always smaller than the entry of the column "GMRes". Row 1 in Table 4.1 shows that for a single grid computation with 60 GMRes iterations the linear solving procedure takes approximately half the time of the nonlinear iteration. The other half is used for the assembling of the Jacobian. When using the Galerkin transfer the time invested in the Jacobian is nearly constant for all multigrid algorithms, see the last column of Table 4.1. Hence, for the solver algorithms proposed in this thesis the main work is conducted in linear algebra operations, since all other timings in the table grow when multigrid algorithms are applied.

The column "cycle" in Table 4.1 shows the growth in computing time if a nonlinear h-multigrid algorithm with a Backward–Euler smoother including a h-multigrid preconditioned GMRes method is applied per iteration instead of a single-grid Backward–Euler algorithm from row 1. This increase in computing time per cycle in comparison to a single-grid solver must be compensated through an overall improved convergence behavior in the solution process in order to achieve an overall faster algorithm in terms of CPU time. Irrespective of theoretical properties an overall gain in computational efficiency

will be demonstrated in Chapter 5 for practical cases by applying the proposed multigrid strategies.

4.9 Solver overview

This section describes the overall solution algorithm and possible solver choices. Figure 4.7 gives an overview of the framework which can represent a single level solver with no multigrid algorithm applied, a solver using a linear multigrid, a nonlinear multigrid solver, and the combination of a nonlinear and a linear multigrid algorithm both for h and p.

Starting from a nonlinear problem formulation the nonlinear multigrid **Algorithm 1** is used to solve this problem, Figure 4.7 (top left corner). The nonlinear multigrid performs a multigrid cycle in each iteration. This cycle is represented in Figure 4.7 (top right corner) as a three level V-cycle. Every other known cycle can be chosen and applied with any desired depth. Even a single grid computation can be simulated by choosing only one level as depth. Now, a nonlinear problem needs to be solved on every level of the nonlinear multigrid cycle. Hence, the Backward–Euler **Algorithm 2** is applied as a nonlinear smoother/solver on every level, as mentioned in Section 4.2. For example, in Figure 4.7, three separate Backward–Euler algorithms are used, one on every level of the nonlinear V-cycle. A flexible implementation allows a different number of Backward–Euler iterations on each level.

Within every Backward–Euler iteration, a linear problem needs to be solved. In this framework, it is possible to apply two different approaches to solve the linear problems. One choice is to bypass the GMRes method and to use the linear multigrid **Algorithm 3** as a linear solver. The other choice is to apply a GMRes method to the linear problems resulting from the Backward–Euler algorithm. In the GMRes method the linear multigrid is applied as a preconditioner in every GMRes iteration. Hence, both linear solver strategies use the linear multigrid algorithm, see Figure 4.7 (bottom right corner). Moreover, the number of GMRes iterations or linear iterations and preconditioner steps depend on the solver settings.

The linear multigrid algorithm performs a multigrid cycle in each iteration step. This cycle is represented in Figure 4.7 (bottom left corner) as a V-cycle. The depth of the linear multigrid cycle depends on which nonlinear multigrid level the Backward-Euler algorithm was executed. For example, if the Backward–Euler algorithm was called on the mid level of the three level nonlinear V-cycle, the linear multigrid would perform a two level cycle. Hence, the linear multigrid uses only coarse levels from the nonlinear multigrid below the current operating level if both multigrid algorithms are combined. If the linear cycle depth can not be reached because there are not enough coarse levels available on the current

operating level, then the linear multigrid algorithm will perform a cycle with the maximal available depth. The linear multigrid algorithm can also run as a single grid linear solver if the cycle depth is set to one. If a nonlinear multigrid cycle with depth greater than one is applied the linear cycle depth can only vary between one level and the nonlinear multigrid cycle depth. On every level of the linear multigrid cycle the Line-Jacobi **Algorithm 4** is applied.

Note that, no startup strategy is included in the overview shown in Figure 4.7. However, the described framework can be employed on every level of the startup strategy.

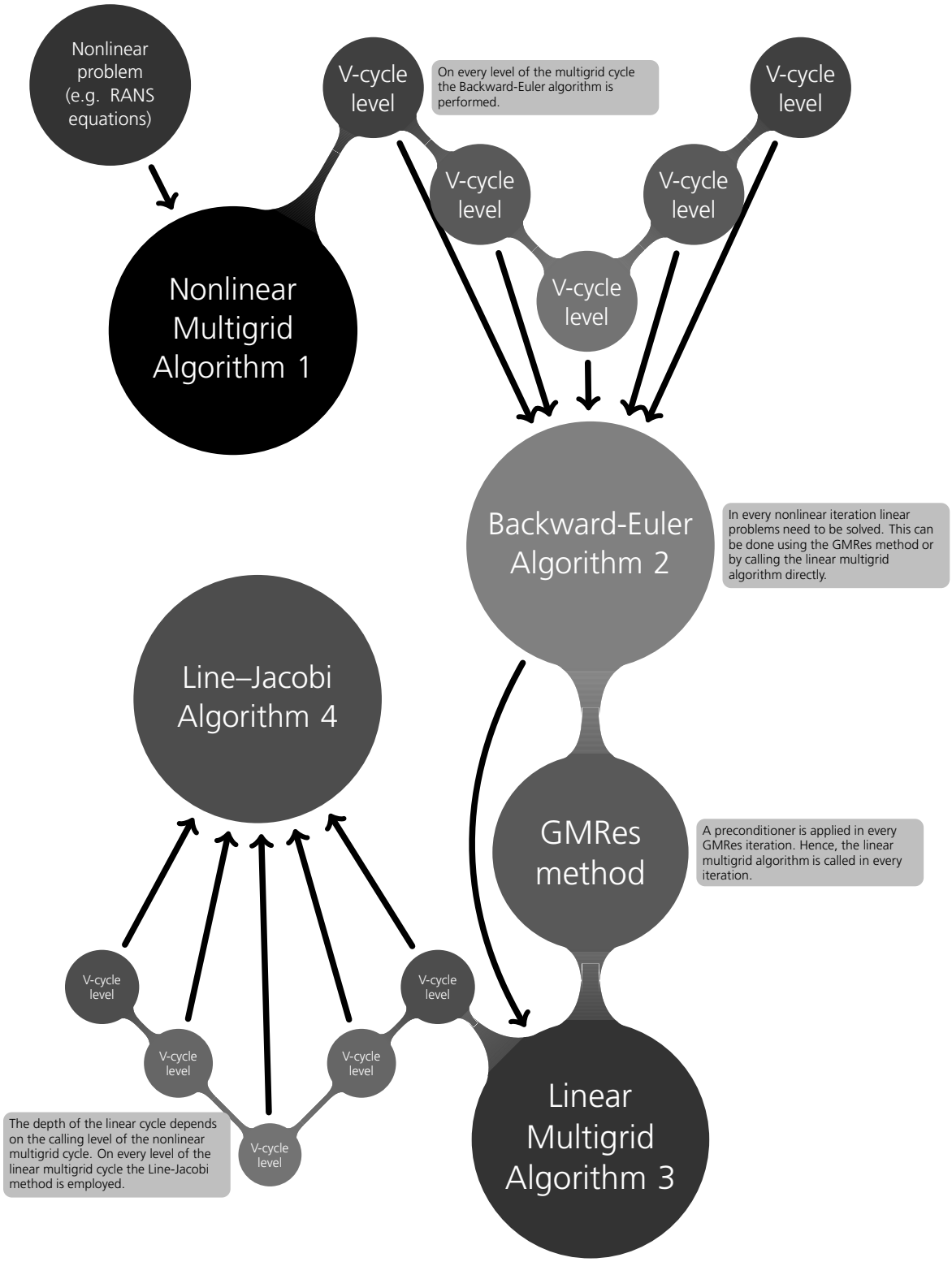

Figure 4.7: Overview of the solver algorithms implemented in the framework.

Now, let n be the number of levels in the nonlinear multigrid algorithm, hence $n = l_{\text{max}} - l_{\text{min}} + 1$. Assuming that the same number k_{NMG} of pre- and post-smoother steps are performed on every level in the nonlinear multigrid algorithm, with $m_{1,l} = m_{2,l} = k_{\text{NMG}},\ l \leq n,\ k_{\text{NMG}} \in \mathbb{N}$, in **Algorithm 1** and assuming that the same number of GMRes iterations i_{GMRes} are conducted in every smoother step, then the total number of linear GMRes iterations per nonlinear multigrid cycle can be computed via,

$$\begin{array}{ll} (2n-1)\, i_{\text{GMRes}}\, k_{\text{NMG}} & \text{for a V-cycle,} \\ (2^n-1)\, i_{\text{GMRes}}\, k_{\text{NMG}} & \text{for a W-cycle.} \end{array} \tag{4.9}$$

Clearly, this formula can also be applied for V-cycles and W-cycles in the linear multigrid algorithm, where k_{LMG} would be the number of linear smoother steps and i_{cycle} the number of performed cycles. Then the number of Line–Jacobi iterations per nonlinear V-cycle is given by

$$i_{\text{GMRes}}\, k_{\text{NMG}}\, i_{\text{cycle}}\, k_{\text{LMG}} + \sum_{j=2}^{n} (2j-1)\, i_{\text{cycle}}\, k_{\text{LMG}}\, 2\, i_{\text{GMRes}}\, k_{\text{NMG}} \tag{4.10}$$

The formula (4.10) holds under the assumption that the same number k_{NMG} of smoother steps are performed on every level in the nonlinear multigrid algorithm V-cycle and the same number of GMRes iterations i_{GMRes} are conducted in every smoother step. Moreover, in every GMRes iteration a constant number i_{cycle} of linear multigrid V-cycle is performed with the same number k_{LMG} of smoother steps on every level, with $m_{1,l} = m_{2,l} = k_{\text{LMG}},\ l \leq n,\ k_{\text{LMG}} \in \mathbb{N}$, in **Algorithm 3**.

The first term of formula (4.10) computes the Line–Jacobi iterations if the nonlinear multigrid V-cycle performs smoother steps on the level l_{min}. The sum computes the Line–Jacobi iterations on all the other levels. The first three factors in the sum are derived from (4.9) since in every GMRes iteration a fixed number of linear V-cycles is performed with $(2j-1)$ levels depending on the level $l \neq l_{\text{min}}$ where the nonlinear multigrid algorithm performs smoother steps. Since all levels in the sum are visited two times and every time k_{NMG} nonlinear smoother steps are performed including i_{GMRes} GMRes iterations in every smoother step the last three factors of each summand are derived. In (4.10) the factors $i_{\text{GMRes}}\, k_{\text{NMG}}\, i_{\text{cycle}}\, k_{\text{LMG}}$ can be factorized and the remaining terms summarized, leading to

$$(2n^2-1) i_{\text{GMRes}}\, k_{\text{NMG}}\, i_{\text{cycle}}\, k_{\text{LMG}}.$$

Since two algorithms apply V-cycle and one of them is used in every smoother step of the other the quadratic ratio over the number of levels in the equation can be seen in this formulation. Moreover, this formulation can be derived from (4.9).

If a three level V-cycle is performed by the nonlinear multigrid algorithm with 1 Backward–Euler smoother step on every level and 30 GMRes iterations in the Backward–Euler iteration, then 150 GMRes iterations are performed per nonlinear multigrid cycle. Furthermore, if only 1 linear multigrid cycle is applied as a preconditioner in every GMRes iteration, then also 150 linear multigrid cycles are conducted per nonlinear multigrid cycle. Moreover, if for the linear multigrid algorithm 1 V-cycle is performed with 4 smoother steps conducted every time the linear multigrid V-cycle applies the smoother then 2040 Line–Jacobi iterations are performed in every nonlinear multigrid V-cycle, calculated with formula (4.10).

The number of Line–Jacobi iterations per level l under the same assumptions for a V-cycles can be computed via,

$$\begin{cases} 2^2\,(n-j+1)\,i_{\text{GMRes}}\,k_{\text{NMG}}\,i_{\text{cycle}}\,k_{\text{LMG}} & \text{for } 2 \leq j \leq n, \\ (2n-1)\,i_{\text{GMRes}}\,k_{\text{NMG}}\,i_{\text{cycle}}\,k_{\text{LMG}} & \text{for } l_{\text{min}}. \end{cases}$$

For the previous example the Line–Jacobi iterations per level l are 600 on l_{min}, 960 on the mid level and 480 on l_{max}. It follows from the given formula that the most smoother steps are conducted on the coarsest level but level l_{min} if the same numbers k_{NMG} and k_{LMG} are employed on every level. While here only the number of iterations are considered it should be noted that iterations on lower levels of a multigrid algorithm can be conducted faster and with less cost, see Table 4.2 and Section 4.8.

Note that in the following only the cost on lower levels of a nonlinear multigrid cycle or startup strategys is addressed. How the cost scales within one computation is the sole purpose of the following investigation. An investigation of grid size in correlation to CPU time on a mesh hierarchy for full CFD computations of the current framework is given in Section 5.2.9. The L1T2 three element airfoil test case and mesh applied in Table 4.2 is introduced in detail in Section 5.1.

no.	level	DoF	elements	degree	CPU time			matrix entries
					All	GMRes	Jac.	
1.	top	142 922	23 824	2	1.00	0.29	0.65	3 695 040
2.	mid	38 904	6 484	2	0.55	0.09	0.44	1 328 832
3.	low	10 890	1 815	2	0.36	0.03	0.32	415 836
4.	top	238 240	23 824	3	1.00	0.30	0.63	10 262 000
5.	mid	142 922	23 824	2	0.46	0.12	0.33	3 695 040
6.	low	71 472	23 824	1	0.21	0.04	0.16	923 760

Table 4.2: Nonlinear iteration cost analysis on agglomerates within the current framework: turbulent L1T2 three element airfoil test case on an unstructured mesh.

In every row of Table 4.2 one Backward–Euler step with 30 GMRes iterations and a 4 iteration Line–Jacobi preconditioner is performed, including the assembling of the Jacobian

on the specific level of the Backward–Euler iteration. The column "CPU time" includes the time of such a nonlinear iteration on the given level normalized with the CPU time on the associated top level under "All" in that column. Under "GMRes" the normalized CPU time spent in the GMRes algorithm is stated and under "Jac." the time spent assembling the Jacobian on the specific level is shown. The timings "GMRes" and "Jac." represent the main operations in one iteration, hence, adding these normalized CPU times should always be close to the time "All" in the specific row. For the rows 1-3 the CPU times are normalized with the time "All" form row 1 and for the rows 4-6 the CPU times are normalized with the time "All" from row 4. The first three rows represent one nonlinear iteration on agglomerations and on the fine mesh for polynomial degree $p = 2$. In this case the agglomeration coarsening rate of the mesh is chosen to be 4, the agglomerates are shown in Figure 4.6. The last three rows represent one nonlinear iteration for different polynomial degrees with the top level of polynomial degree $p = 3$ and the associated quadrature points of the top level. Note that, during integration all quadrature points from the associated top level are considered for both multigrid and startup strategy types. For the first three rows of Table 4.2 the overall CPU time "All" does not scale in a h-multigrid sense with the same factor of the applied coarsening rate like the elements do. In contrast to that the last three rows the overall CPU time "All" scale with a factor of ~ 2, since a p-hierarchy is investigated and the scaling factor of the time "Jac." for the Jacobian operations is the dominant factor. Moreover, for these rows the time for "GMRes" iterations scales similar to the matrix entries for each level.

For the lower levels the quadrature points from the top level are applied. This is possibly a significantly larger number of quadrature points than would be required on that level. However, the difference between the number of required and applied quadrature points in the last three rows is not as large as in the agglomeration based computations in the top three rows of Table 4.2. This explains the different scaling factor of the CPU time in the column "Jac." between the first three rows and the last three rows. For the agglomeration based computation h-hierarchy in Table 4.2, with the applied coarsening rate of 4, the number of quadrature points per element are increased by a factor of ~ 16 between the low and top level. Hence, for the h-hierarchy case the number of quadrature points per lower level increase multiplied by the coarsening rate applied. In contrast to that in the p-hierarchy the number of quadrature points per element is increased by a factor of ~ 3 between the low and top level only. Hence, in the p-hierarchy case the number of applied quadrature points on one element in comparison to the required quadrature points does not differ as much as for the h-hierarchy case. Thus, the time spent in the Jacobian "Jac." does not scale in a multigrid sense over the number of degrees of freedom (DoF) for the top three rows of Table 4.2, as mentioned before in Section 4.8. Applying the Galerkin transfer for h-multigrid cycles seems to be an appropriate approach to tackle this issue, since the Galerkin transfer replaces the need for reassembling the Jacobian

on lower levels. Hence, the Galerkin transfer was not applied in Table 4.2. A satisfying reduction in CPU time is due to the reduced size of vectors and matrices involved since linear algebra operations are perfromed faster on fewer elements. Hence, the overall "GMRes" time scales like the number of matrix entries in the top three rows of Table 4.2. Table 4.2 shows that the Backward–Euler iterations, including the assembling of the Jacobian, performed on lower levels of the multigird algorithms or startup strategies can be conducted faster in comparsion to the same nonlinear iteration performed on the top level. Since the system matrix on each level takes up a major part of the required memory the last column states the number of entries of the system matrix which need to be stored on each level. It follows that on lower levels less memory is needed and the top level consumes the major part of the overall memory required.

Pseudo-time steps on multigrid algorithm levels

Within Algorithm 2 the a pseudo-time stepping scheme $(\mathrm{CFL}_i \Delta t_0)$, based on local time steps computed from the local state and a given CFL number is applied. However, there are two possible ways to obtain the time step on the lower levels of the multigrid algorithms. One way is to transfer the time step from the top level, the other one is to recompute the time step on every level. The approach of transferring the time step is reasonable in the case of a linear multigrid algorithm, in which the same linear problem including the fine level pseudo-time stabilization term is to be approximated on each level. Therefore for all sole linear multigrid computations the approach of transferring the time step is chosen. In contrast to that, the time step is recomputed on coarse levels in the case of a nonlinear multigrid algorithm. Based on the same CFL number but larger agglomerated elements the resulting time step is larger on coarser levels, yielding a better progress in pseudo-time on the coarse level.

In some of the computations, both the nonlinear multigrid algorithm as solver and the linear multigrid algorithm as a preconditioner or linear solver are employed. In order to avoid the construction of different linearized operators for the linear and nonlinear components in the case of this combined algorithm, a single approach to determine a time step is chosen. Numerical experiments have shown that the results obtained with a projected fine level time step are inferior to those obtained with a recomputed time step size when combining both, linear and nonlinear h-multigrid algorithms. Please note that, in the case of a p-multigrid algorithm the mesh stays the same on all levels, thus the original fine level time step is identical to both, the projected and the recomputed value, thus eliminating the need to choose in that case.

Matrix-free implementation and parallelization

Nearly every part of this framework can be implemented in a matrix-free fashion since all algorithms involved mostly require matrix–vector products , which can be implemented matrix-free using finite differences. However, saving memory through a matrix-free ap-

proach mostly results in an increased overall CPU time per computation for these algorithms as previously mentioned in Section 4.4. In addition to that, the finite differences used in such an approach always need some parameter handling as well, i.e. the determination of an optimal step size.

Here, a matrix-based implementation in combination with a suitable parallelization is the desired approach for this framework. The mesh is decomposed into subdomains and each of which is assigned to a processor. Such a domain-decompositioned parallelization holds parts of the mesh per process only. All parts combined reassemble the initial mesh. Then all global matrices and vectors employed by the solver algorithms have to be computed and stored for the mesh part of the specific subdomain. Hence, the memory used by a sequential computation can be uniformly distributed across the processors of a parallel computation, resolving possible memory issues. However, an overlapping communication layer is needed where information, e.g. for numerical flux functions, is transferred between domains. The implementation of such a communication layer is similar to the idea of introducing a ghost layer of elements at the domain boundaries, which was also described in Section 3.3. This communication layer is only one mesh element wide because DG methods always have a compact stencil, transfering data between neighbor elements only, for every polynomial degree and therefore order of the method, as seen in Chapter 3. Hence, the memory overhead of the parallelization is quit small, since it is a ghost layer of boundary elements per domain and a vector that holds the information where to find the neighbor elements of a domain boundary.

The domain-decomposition is realized in such a way that the agglomeration stays the same for the same mesh if the same coarsening rate c_{rt} is chosen, no matter how many processors are used. This is due to the fact that the domain-decomposition takes place on the coarsest mesh and is transferred to the initial fine mesh elements afterwards.

This domain-decompositioned parallelization approach resolves the possible memory issues, since up to this point none could be observed within this framework on a modern cluster system. Without being penalized with a large increase in CPU time this approach is the preferred option in comparison to the matrix-free approach. If memory issues occur a matrix-free approach on every domain could further address the memory issue.

In Table 4.3 a three level h-startup strategy, **Algorithm 5**, is applied with a startup tolerance of $r_{\mathrm{tol}} = 10^{-8}$. Moreover, the complete framework is employed including a nonlinear h-multigrid algorithm with a Backward–Euler smoother and a linear h-multigrid preconditioned GMRes method. The column "nonlinear V-cycle" in Table 4.3 includes the number of nonlinear V-cycles required starting on the lowest level of the startup strategy up to the top level for all $p = 2$ computations. The test case and mesh applied in Table 4.3 is introduced in detail in Section 5.1. Note that, on the lowest level of a startup strategy a V-cycle is equal to one single grid Backward–Euler iteration, since no lower levels are

available. For this test cases the difference in the number of nonlinear V-cycles required for a varied number of processors was at most two, which could be explained via round off errors in the solution process and small differences in the line sets created, as described before.

no.	domains	CPU time	optimal time	nonlinear V-cycle
1.	1	1	1	21, 19, 56
2.	2	0.49	0.5	21, 19, 54
3.	4	0.26	0.25	21, 19, 54
4.	8	0.147	0.125	21, 19, 55
5.	16	0.0873	0.0625	21, 19, 56

Table 4.3: Domain decomposition parallelization effect on current framework: turbulent L1T2 three element airfoil test case with a $p = 2$ computation on an unstructured mesh.

Most of the algorithms involved cope fine under parallelization. Therefore, a performance decrease is not likely, see Table 4.3. The only algorithm which may not fulfill his full potential is the Line–Jacobi algorithm in case lines are cut by subdomain boundaries. However, only a few couplings are lost if a lines is cut. Therefore, not a large reduction in the overall performance is expected [76]. Since the GMRes method is stopped only if either a certain linear residual reduction is reached or the maximal number of iterations is performed, the potential deterioration caused by the Line–Jacobi algorithm could only appear if the maximal number of iterations is performed. The maximal number of GMRes iterations is mostly reached at the end of the nonlinear solution process where the regime of asymptotic convergence is reached and therefore, influences the convergence rate at the end of the solution process only. This argument is supported by the last column in Table 4.3, where a difference in the number of nonlinear V-cycles is noted at the top level of the startup strategy only where the nonlinear residual is reduced to 10^{-12}. However, no significant deterioration in the convergence performance could be observed.

Since the agglomeration stays the same for every given number of subdomains the lines on every level are cut over domain borders, thus weakening the Line–Jacobi method. Every entry in the column "CPU time" was normalized with the CPU time of the computation shown in Table 4.3 (row 1). The column "optimal time" shows entries of the normalized CPU time if the code would scale perfectly. The differences between the entries in the columns "CPU time' and "optimal time" are very small and therefore, the results are quite satisfying. Most of the difference is explained by the one or two nonlinear iteration difference on the top level of the startup strategy and communication as well as sequential code overhead. For the results shown in Table 4.3 (row 5) each of the 16 domains hold approximately 110 coarse level mesh elements per domain. This should serve as a guide value while choosing the agglomeration coarsening rate and therefore, the number of elements of the coarsest agglomerate depending on number of processors used.

5 Computational test cases

At the beginning of this section, all test cases including the computational meshes are introduced, which are referred to when numerical investigations of the proposed solver algorithms are performed. Note that all test cases considered are well known and investigated in the field of Computational Fluid Dynamics (CFD). In Section 5.1, a validation is given of the framework described in Chapter 4 comparing flow solutions against experimental data and reference values from other CFD codes in terms of accuracy. In Section 5.2, investigations and comparisons between different solver choices and settings within the suggested framework are discussed. An important point is that for all the investigated phenomena of the proposed solver algorithms a test case independency can be shown. Therefore, all final results are legitimated for all test cases considered and could be transferred to similar configurations. At last, some test cases are revisited in Section 5.3 and a comparison of the proposed solver algorithms with other CFD codes in terms of work units (cf. Appendix B) is shown.

Furthermore, when results are compared with other CFD codes, the framework described in this work is referred to as DG_ON.

5.1 Test case introduction and framework verification

At first, two three element airfoils in a high lift configuration are introduced. Although these two test cases differ in geometrical details only, the meshes for these test cases are significantly different. For one high lift configuration, namely the L1T2, two unstructured grids are provided, whereas for the MDA 30P30N a nested hierarchy of four structured meshes is available. At first a p-enhancement study is performed, inorder to show a increase in accuracy for C_L and C_D for higher polynomial degrees. For the second test case, namely the MDA 30P30N, a mesh convergence study for a fixed polynomial degree is performed. Both of these study are necessary as a verification for the proposed framework DG_ON.

As a three dimensional test case, the VFE-2 delta wing with rounded leading edge and the NASA Trap Wing is considered. For all test cases, CFD data from several codes are

available including finite volume (FV) and discontinuous Galerkin (DG) results, which are used for validation purposes.

5.1.1 L1T2 high-lift three element airfoil

The first test case considered is a flow around a high-lift three element airfoil with blunt trailing edges. The L1T2 test case is defined by the following flow conditions:

- Mach number M $= 0.197$,
- Reynolds number Re $= 3.52 \cdot 10^6$,
- angle of attack $\alpha = 20.18°$.

This case has been documented in detail in [21]. Therefore, additional data are available to verify the algorithms in terms of accuracy with respect to force coefficients. Most of the data which are used for comparison are taken from the EC funded projects ADIGMA [34] and IDIHOM [33].

In this section, higher order computations on two unstructured meshes will be shown. Both meshes are shown in Figure 5.2. The mesh on the top of Figure 5.2 has 25 757 elements and the one on the bottom has 23 824 elements. Moreover, the mesh on the bottom of Figure 5.2 and the agglomerates, which are used in the h-multigrid algorithm and startup strategy, can also be seen in Figure 4.6.

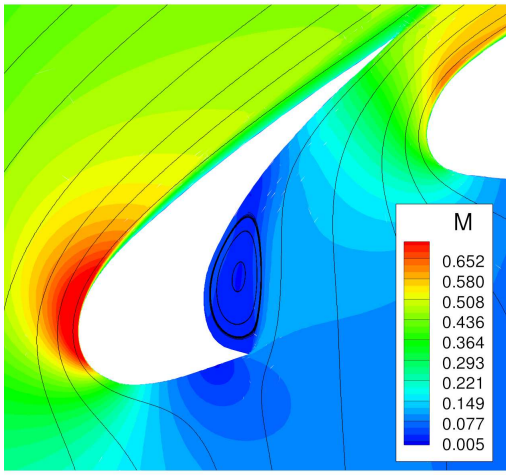

Figure 5.1: Mach number distribution on the L1T2 slat of a third order RANS-$k\omega$ solution on an unstructured 23 824 element mesh.

Note that both meshes in this section have also been used in the IDIHOM project [33]. The mesh described in the upper row of Table 5.1 was provided by the Warsaw University of Technology and will be labeled as WUT mesh. This mesh was generated by adaptively refining a coarse mesh while applying the SA turbulence model. The mesh described in the lower row of Table 5.1 was provided by CENAERO and will be labeled as CEN mesh from here on. This mesh was created with GMSH [24].

In comparison to the WUT mesh, the CEN mesh has less quadrangles while both have nearly the same amount of triangles. However, in the WUT mesh all quadrangles are

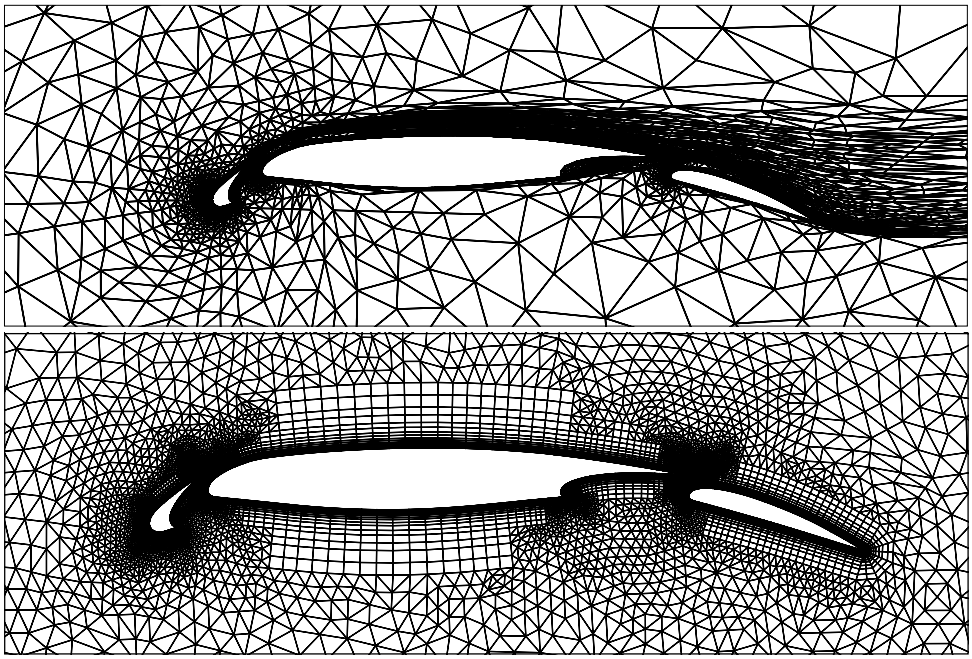

Figure 5.2: Top: Unstructured L1T2 mesh with 25 757 elements (WUT mesh), Bottom: Unstructured L1T2 mesh with 23 824 elements (CEN mesh).

located in the boundary layer, whereas in the CEN mesh quadrangles are located also in the convective dominated area around the main wing and the flap of the airfoil, see Figure 5.2.

name	no. of elements	quadrangles	triangles	DoF (p=2)	comment
WUT mesh	25 757	10 424	15 333	154 542	adaptively refined
CEN mesh	23 824	7 880	15 944	142 944	GMSH

Table 5.1: Test case mesh description.

For a $p = 1$ solution the degrees of freedom (DoF) per equation in 2D are 3 times the number of elements, for $p = 2$ its 6 times and for $p = 3$ its 10 times the number of elements, resulting from (3.23). For comparison, in cell-centered FV methods the number of DoF are equal to the number of mesh elements.

C_L	C_D	turbulence model	code	discretization
3.9610	0.0710	$k\omega$	PADGE	DG
4.0510	0.0608	SA	TAU	FV

Table 5.2: Reference values for the lift and drag coefficients of the L1T2 test case.

The first reference values in Table 5.2 (row 1) are taken from computations in [26] with the DLR Parallel Adaptive Discontinuous Galerkin Environment (PADGE) [27] on an adjoint-based refined mesh originated from a structured 4 740 element mesh. Please note that

these are reference values computed with the $k\omega$ turbulence model in a DG environment and should be used for comparison with results obtained with computations where the $k\omega$ turbulence model is used.

Moreover, the DLR has provided reference solutions in the IDIHOM project with the unstructured node-centered FV TAU code [41] for the SA turbulence model. Computations were performed on a mesh hierarchy of four structured meshes from 4 268 elements up to 273 152 elements. The proposed evaluation according to the IDIHOM evaluation procedure on this hierarchy of structured meshes yields asymptotic values of the lift and drag coefficient, which can be seen in Table 5.2 (row 2). These asymptotic values of the FV data are used for the computation of errors in the DG case, as no mesh convergence study could be done on the two available unstructured meshes. Here, the same error threshold is used as defined in the IDIHOM project, see Figure 5.3. The actual asymptotic values might be slightly off due to subtly differences in the airfoil geometry approximation as well as the location and treatment of far field boundaries.

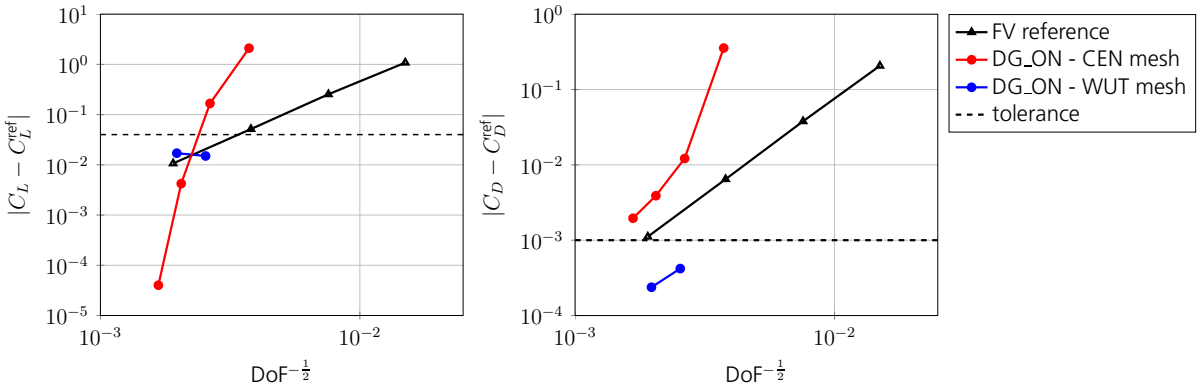

Figure 5.3: Mesh convergence of error in lift (left) and drag coefficients (right) for reference computations and DG results.

The two dots for the DG results on the WUT mesh in Figure 5.3 represent third ($p = 2$) and fourth ($p = 3$) order Reynolds-Averaged Navier-Stokes (RANS)-SA solutions. The values for the lift and drag coefficient on the unstructured WUT mesh for the fourth order DG method are $C_L = 4.03393$ for the lift coefficient and $C_D = 0.0610$ for the drag coefficient. The four dots in Figure 5.3 for the unstructured CEN mesh show DG results from second order ($p = 1$) up to fifth order ($p = 4$) RANS-SA solutions. Furthermore, for both higher order meshes the tolerance criterion is meat for C_L if the order is chosen sufficiently high. The unstructured WUT mesh, which was created by refining a coarse mesh using the SA turbulence model, seems to fulfill the tolerance criterion better overall for both values. The CEN mesh does not resolve all relevant flow field characteristics sufficiently well for both target values. Moreover, only with a polynomial degree of at least 3 a satisfying C_L value is obtained, whereas for smaller polynomial degrees critical flow field regions are not sufficiently resolved. Furthermore, the drag coefficient C_D does not meet the

criterion, possibly due to an under resolution of the boundary layer for the CEN mesh. In comparison to the WUT mesh where the maximum thickness of the first element layer on the boundary is $1.45\,10^{-5}$ the maximum thickness on the CEN mesh is $1.35\,10^{-4}$. A similar difference exists also for the minimum thickness values of both meshes. Thereby increasing the polynomial degree on the CEN mesh might not be sufficient to conquer this issue for C_D.

However, the overall results are very satisfying since they show that for the proposed framework a convergence study by enhancing the polynomial degree, and therefore the DoF, on a given mesh will increase the accuracy of the resulting flow solutions in terms of C_L and C_D in comparison with asymptotic values computed with a FV code and the same turbulence model. This result is therefore a verification of the given framework.

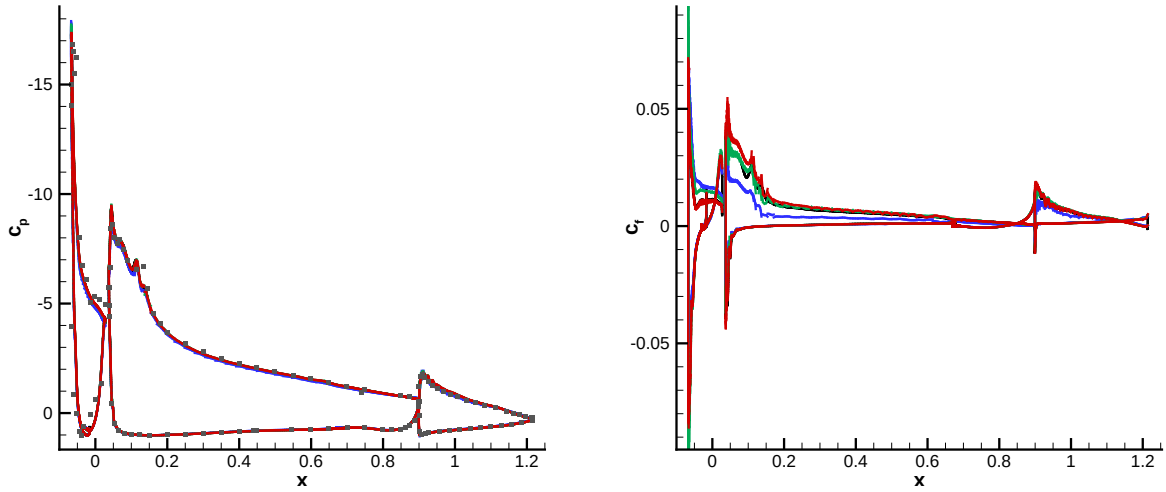

Figure 5.4: Comparison between FV RANS-SA reference solution (black) and results for the RANS-SA equations for $p = 2$ (blue), $p = 3$ (green) and $p = 4$ (red) on the CEN mesh in terms of computed pressure coefficient (left) and skin friction coefficient (right). Symbols indicate experimental data.

Figure 5.4 shows the boundary surface plots of the computed pressure and skin friction coefficient of $p = 2$ (blue), $p = 3$ (green) and $p = 4$ (red) computations on the CEN mesh for the SA turbulence models. For the pressure coefficient c_p, the results are satisfying. The $p = 4$ computation shows the best results in comparison to experimental data and $p = 2$ the worst. This is mostly visible at the leading edge of the slat and main wing. Furthermore, the FV reference solution represented by a black curve is nicely covered by the DG_ON results. For the skin friction coefficient c_f a difference from the reference solution can be observed. At the slat the DG_ON results converge to the reference solution while enhancing the polynomial degree, the $p = 4$ results covers exactly the reference solution on the slat. On the flap all DG_ON results for c_f show the same behavior. For the main wing the $p = 4$ results overpredict the c_f values at the leading edge in comparison to the reference results but show the same characteristics. These results for c_f correspond

to the results shown in Figure 5.3 for the drag coefficient C_D which has not met the tolerance criterion on the CEN mesh.

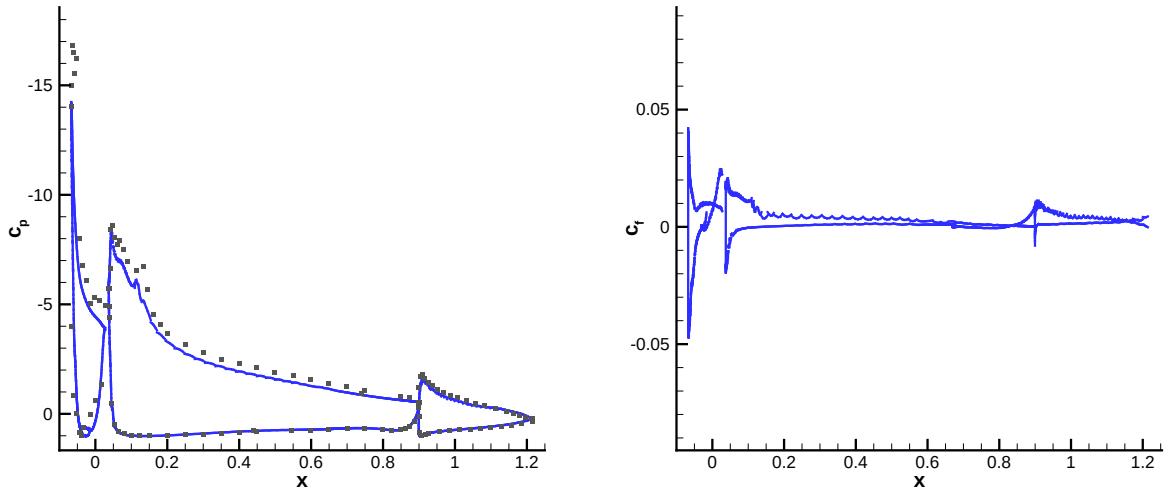

Figure 5.5: Results for the RANS-$k\omega$ equations for $p = 2$ (blue) on the CEN mesh in terms of computed pressure coefficient (left) and skin friction coefficient (right). Symbols indicate experimental data.

For the RANS-$k\omega$ results in Figure 5.5 quite a difference in c_p from the experimental data can be observed, which is due to the mesh being not sufficiently fine for a third order RANS-$k\omega$ computation. Following from Figure 5.3 the same observation was drawn for the $p = 2$ computation in combination with the RANS-SA equations. In terms of the skin friction coefficient no experimental data was available. However, the skin friction coefficient looks similar to the $p = 2$ RANS-SA results shown in 5.4.

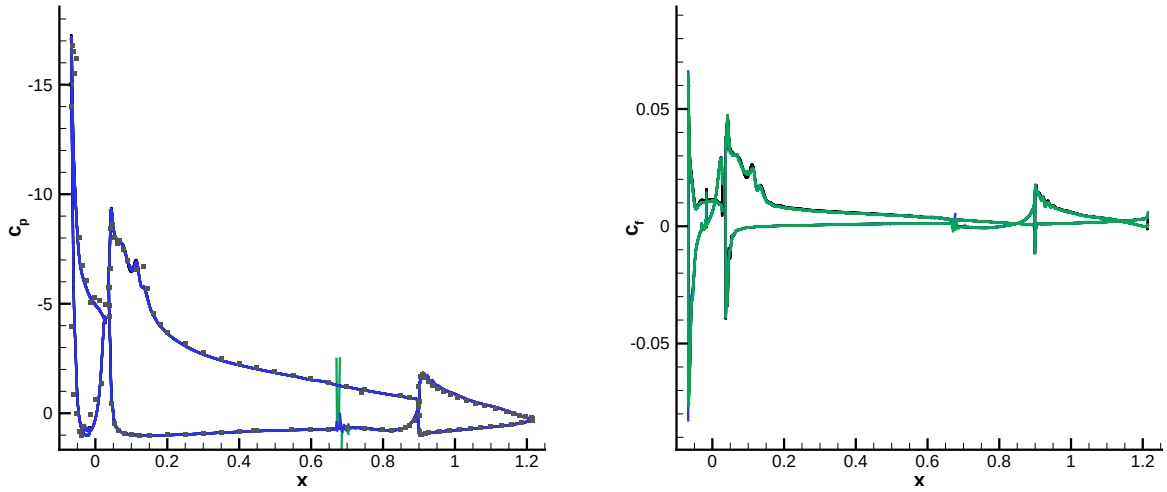

Figure 5.6: Comparison between FV RANS-SA reference solution (black) and results for the RANS-SA equations $p = 2$ (blue) and $p = 3$ (green) on the WUT mesh in terms of computed pressure coefficient (left) and skin friction coefficient (right). Symbols indicate experimental data.

On the WUT mesh, a good agreement between the FV RANS-SA reference results and the third order RANS-SA DG-computations for both the computed pressure and skin friction coefficient is obtained, see Figure 5.6. The only notable discrepancy is a pressure disturbance at the lower edge of the slat cove for the third order computation which might be due to the mesh topology in that region, see Figure 5.2. This discrepancy in c_p increases if the polynomial degree is increased. This could also explain why the lift coefficient for the $p = 3$ solution on the WUT mesh is slightly inferior in comparison to the $p = 2$ results on this mesh, shown in Figure 5.3. The opposite effect can be seen for c_f , here the peaks of the $p = 2$ resultsare slightly higher in comparison to the $p = 3$ results and the reference solution, which also corresponds to the results shown for the drag coefficients in Figure 5.3.

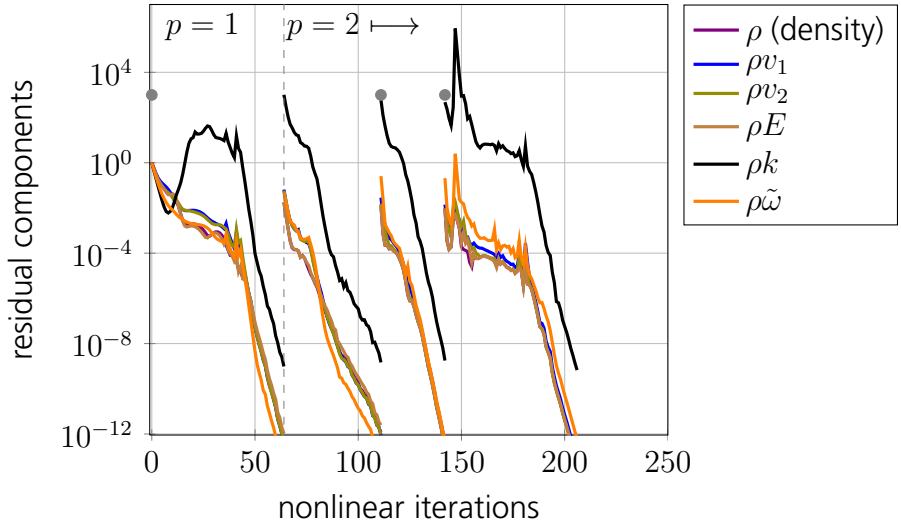

Figure 5.7: Nonlinear convergence of all residual components for a L1T2 $k\omega$-computation at $p = 2$ with a h-startup strategy on the CEN mesh.

Finally, Figure 5.7 shows a convergence plot for the L1T2 test case of a 3 level h-startup strategy with a nonlinear residual reduction of 10^{-12}. The computation was performed on the CEN mesh in combination with the RANS-$k\omega$ equations for a $p = 2$ discretization. At every • a computation is started on the next mesh in the hierarchy of agglomerated meshes shown in Figure 4.6. On the coarsest mesh, a $p = 1$ solution is computed and used as an initial solution for the $p = 2$ computation on the same agglomerated mesh. On the right side of the dashed line in Figure 5.7, $p = 2$ computations are shown on the CEN mesh and both resulting agglomerated meshes. Here, every computed solution on a level is used as an initial solution on the next finer level. Furthermore, Figure 5.7 shows the l^2-norms of all residual components for all computations performed in the h-startup strategy. All residual components Figure 5.7 converge with the same rate. In fact this is true for all computations of this test case on all meshes for both turbulence models shown in Section 5.2 justifying that only the density residual is shown in most of the figures in Section 5.2.

5.1.2 MDA 30P30N three element airfoil

The MDA 30P30N is another high-lift three element airfoil, similar to the L1T2 airfoil considered in Section 5.1.1. The main difference between the MDA 30P30N and the L1T2 airfoil is the shroud with a $90°$ plane angle on the main wing for the MDA 30P30N, as seen in Figure 5.8 in comparison to Figure 5.2. Furthermore, the MDA 30P30N has sharp trailing edges whereas the L1T2 airfoil has blunt trailing edges. Moreover, the MDA 30P30N airfoil is one of the test cases considered in the 2nd International Workshop on High-Order CFD Methods (HOW) [77]. Some of the results of the 2nd workshop are used in this section to verify the results obtained with the DG_ON code.

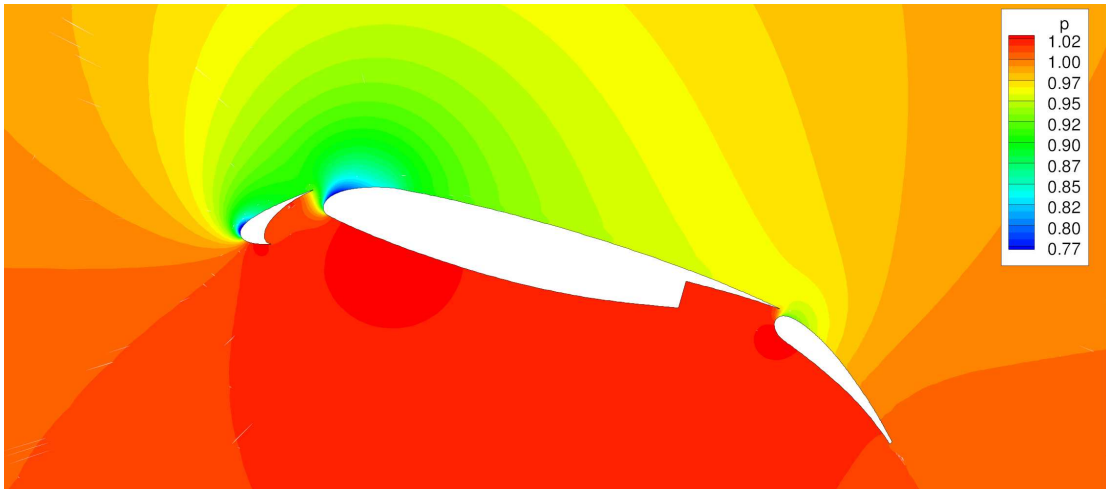

Figure 5.8: Pressure plot of a $p = 2$ solution on a structured mesh with 33 728 elements.

The MDA 30P30N test case is defined by the following flow conditions:

- Mach number M $= 0.2$,
- Reynolds number Re $= 9 \cdot 10^6$,
- angle of attack $\alpha = 16°$.

Computations will be shown on a hierarchy of four higher order quadrilateral meshes from 2 108 elements up to 134 912 elements. The far field distance of these meshes varies between 44 and 50 chord lengths. This hierarchy of higher order meshes has been generated based on a very fine structured quadrilateral FV mesh. The higher order meshes were obtained by coarsening the fine FV mesh and using the additional points to curve the resulting meshes as described in Section 3.5. The second mesh in the resulting hierarchy is shown in Figure 3.2.

Reference values computed with the unstructured node-centered FV TAU code [41] with the SA turbulence model are given in Table 5.3 (row 1). The reference computations were

computed on three different tetrahedral meshes, with 119 510, 240 955 and 485 832 elements.

C_L	C_D	turbulence model	code	discretization	DoF
4.15238	0.04949	SA	TAU	FV	485 832
4.16634	0.04722	SA	DG_ON	DG	809 472

Table 5.3: Reference and DG_ON values for the lift and drag coefficients of the MDA 30P30N test case.

Note that, the same reference values were used at the HOW for this test case. The results from the HOW indicate that most of the participants came close to these values after mesh refinement studies.

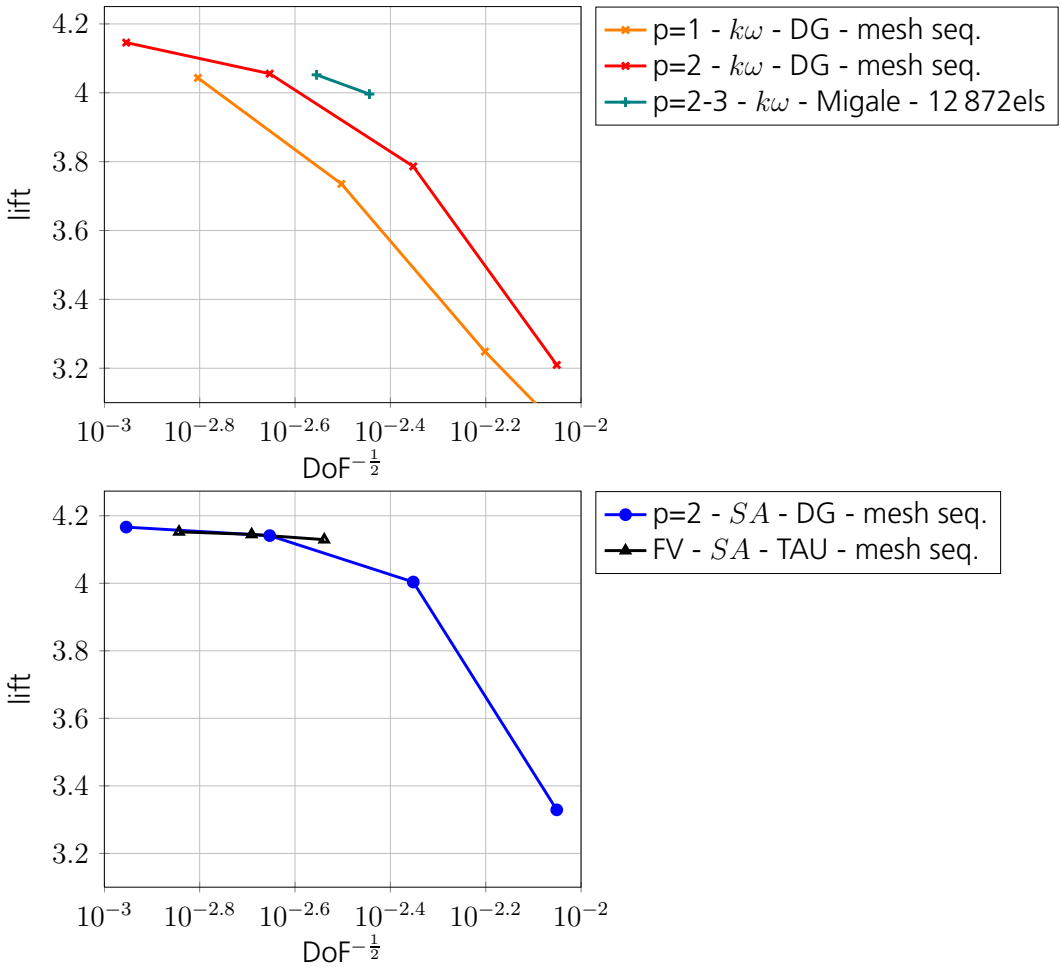

Figure 5.9: Top: Lift of RANS-$k\omega$ eq. in correlation to the mesh size.
Bottom: Lift of RANS-SA eq. in correlation to the mesh size.

Results for the MDA 30P30N test case are presented in Figures 5.9-5.10. The lines marked with "$+$" are results taken from the workshop and provided by the University of Bergamo based on their CFD code Migale. These results represent DG solutions of degree $p = 2$ and $p = 3$ on a structured mesh with 12 872 elements for the RANS-$k\omega$ equations. The

lines marked with "▲" represent the FV results mentioned above on the three tetrahedral meshes. All other lines represent results obtained with the DG_ON code, which is described in Section 4. In particular, lines marked with "•" represent RANS-SA results and lines marked with "×" represent RANS-$k\omega$ results on the mesh hierarchy described above. Note that, the $p = 2$ RANS-$k\omega$ computation from DG_ON on the mesh with 134 912 elements, which are also associated with the red "×" on the far left in Figure 5.11-5.10, is not fully converged due to a solver stagnation at a residual of 10^{-7} equations.

In Figures 5.9 the lift coefficients are plotted over $(\text{DoF})^{-\frac{1}{2}}$. This value $(\text{DoF})^{-\frac{1}{2}}$ is a measure for the mesh size h in 2D. Note that there is a good agreement of the $p = 2$ RANS-SA results with the reference FV results. Also note that, the RANS-$k\omega$ results do not converge to exactly the same reference values as computed with the SA turbulence model. However, the results on the mesh from the University of Bergamo align with the RANS-$k\omega$ results of DG_ON, marked with "•", see Figures 5.9 (top).

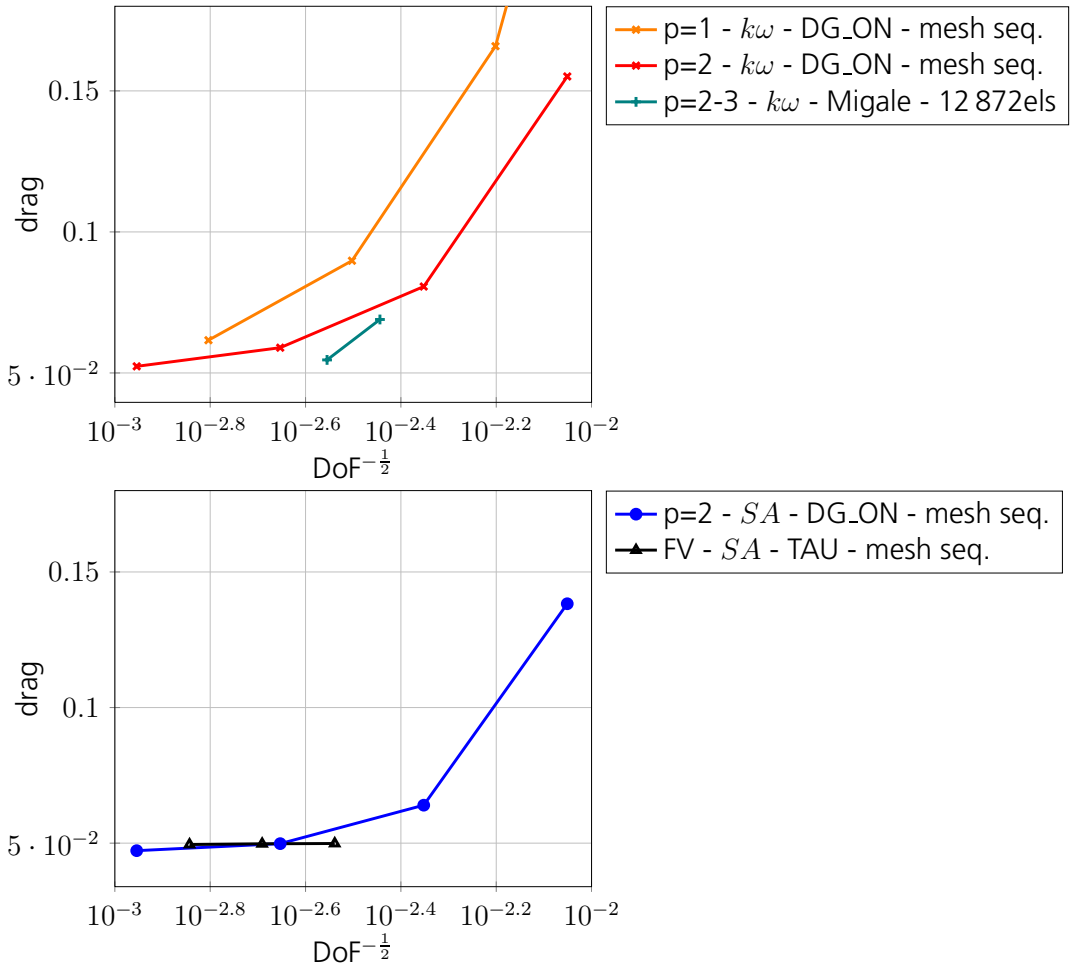

Figure 5.10: Top: Drag of RANS-$k\omega$ eq. in correlation to the mesh size.
Bottom: Drag of RANS-SA eq. in correlation to the mesh size.

Figure 5.10 shows the drag coefficient values of the same computations. The RANS-$k\omega$ results show a small discrepancy in comparison to the the RANS-$k\omega$ results from the

University of Bergamo. All other observations which can be drawn from Figure 5.10 align with those drawn from Figure 5.9 concerning the lift coefficient.

C_L	C_D	turbulence model	code	discretization	DoF
4.05215	0.05462	$k\omega$	Migale	DG	128 720
4.05548	0.05895	$k\omega$	DG_ON	DG	202 368

Table 5.4: RANS-$k\omega$ values for the lift and drag coefficients of the MDA 30P30N test case.

At last the lift and drag coefficient values on the 33 728 element mesh for the $p = 2$ RANS-$k\omega$ computation in comparison to the same values from the Migale $p = 3$ RANS-$k\omega$ computation are collected in Table 5.4.

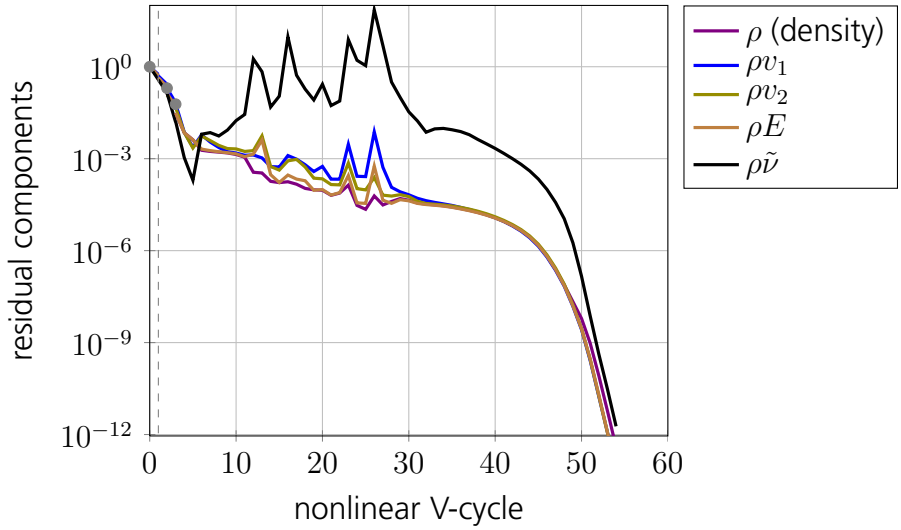

Figure 5.11: Nonlinear convergence of all residual components for an MDA 30P30N SA-computation with $p = 2$ on the 33 728 element mesh.

A convergence plot for the MDA 30P30N test case on the structured mesh with 33 728 elements for a $p = 2$ RANS-SA computation is shown in Figure 5.11. Note that for this computation a one step startup strategy was applied, which performs one iteration on the coarsest agglomerate for $p = 1$ and one $p = 2$ iteration on every agglomerate in order to compute an initial solution for the top level. Moreover, a combination of a nonlinear and linear 3 level h-multigrid algorithm is employed, both using V-cycles. In Figure 5.11 the same display convention is used as for the L1T2 convergence plot in Figure 5.7. At every • a computation is started on the next mesh in the hierarchy of agglomerated meshes. Furthermore, a dashed line indicates an increased p. Figure 5.11 includes all residual components of the RANS-SA equations. Like in the previous section all residual components converge with the same rate and in a satisfying manner. Note that this is the case for all computations of this test case shown in Section 5.2 on all meshes as well as for both turbulence models. Therefore, in Section 5.2 the plots will include

the density residual only, whereas the other components can be expected to behave the same.

5.1.3 VFE-2 delta-wing

The VFE-2 delta-wing with rounded leading edge is a test case considered in the EC funded IDIHOM project [33].

This test case is considered in order to test the applicability of the Line–Jacobi method, **Algorithm 4**, also in 3D. Note, that for a 2D quadrilateral mesh the Line–Jacobi method takes into account half of the Jacobian off-diagonal blocks whereas only a third of the Jacobian off-diagonal blocks is taken into account for a 3D hexahedron mesh. The results will in Section 5.2 indicate that lines can be used for preconditioning also in 3D, even though the number of discarded couplings in comparison to the complete Jacobian is higher than in 2D.

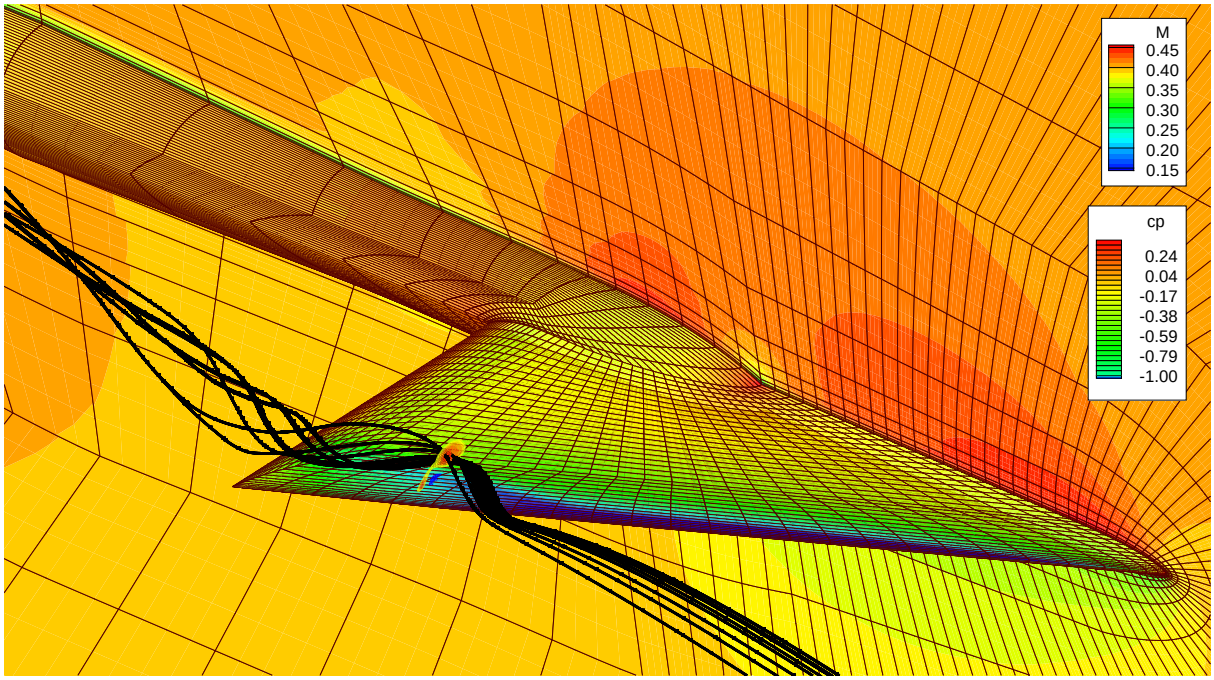

Figure 5.12: Pressure contours on the wing and Mach number on the symmetry plane and streamline plot of a $p = 2$ solution on the mesh with 110 528 elements and experimental data represented in the slice of vorticity.

In Figure 5.12 the surface pressure on the wing is shown for a $p = 2$ solution of the RANS-$k\omega$ equations on a structured mesh with 110 528 elements. The streamlines in this picture are plotted where the primary vortex from the numerical results separates from the wing. The slice in Figure 5.12 represents the vorticity at 80% of the wing chord length extracted from experimental data. The vortex streamlines and the vorticity show an agreement between experimental data and the numerical solution in this area of the flow field.

This test case is defined by the following flow conditions:

- Mach number M $= 0.4$,
- Reynolds number Re $= 3 \cdot 10^6$,
- angle of attack $\alpha = 13.3°$,
- side slip angle $\beta = 0°$.

Computations will be shown on a hierarchy of three structured hexahedral meshes with 13 816 elements up to 884 224 elements. This mesh hierarchy is created using global mesh refinement. For a $p = 1$ solution the number of DoF per equation in 3D is four times the number of elements, for $p = 2$ it is ten times the number of elements. Hence on the mesh with 110 528 elements every equation is discretized with 1 105 280 DoF.

In Figure 5.13 several numerical results for the lift and drag coefficients are shown. All results are computed for the RANS-$k\omega$ equations.

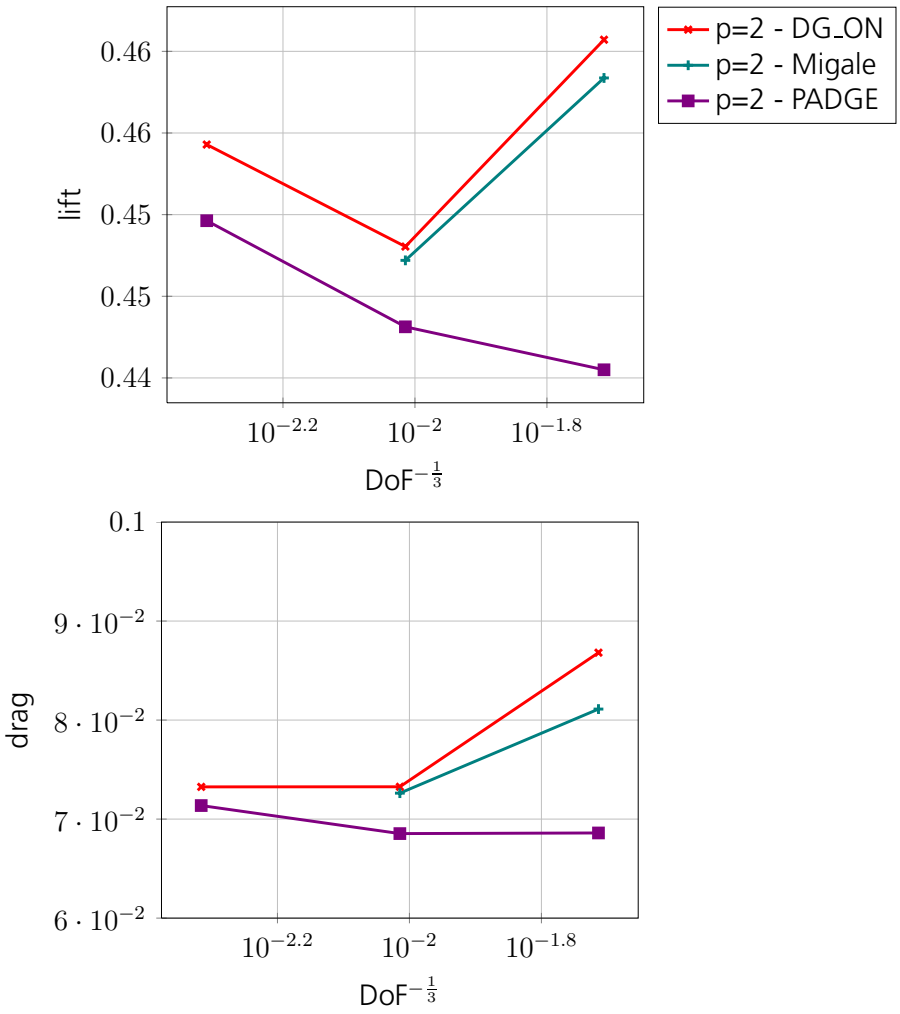

Figure 5.13: Subsonic flow around the VFE-2 delta wing: Convergence of lift (top) and drag (bottom) in correlation to the mesh size.

All results shown are DG results for a $p = 2$ discretization. The teal lines marked with a "$+$" are computed by the University of Bergamo using the CFD code Migale on the first two meshes of the globally refined mesh hierarchy. The red lines marked with a "$\times$" are computed with DG_ON. DG_ON and Migale both use a basis constructed in the physical space, hence, both results for the first two meshes of the globally refined mesh hierarchy are very similar for the lift and drag coefficients. The purple lines marked with a "■" are results computed with the PADGE code. There is a significant difference between the results of the PADGE code and the results from Migale and DG_ON on the coarsest mesh, due to difference in the discretization and basis. For example the PADGE code uses parametric elements, i.e. the basis functions are constructed on reference elements and are then transformed to the cells in the physical space. However, the results on the next finer globally refined meshes are much closer together for the lift and drag coefficients as it is expected.

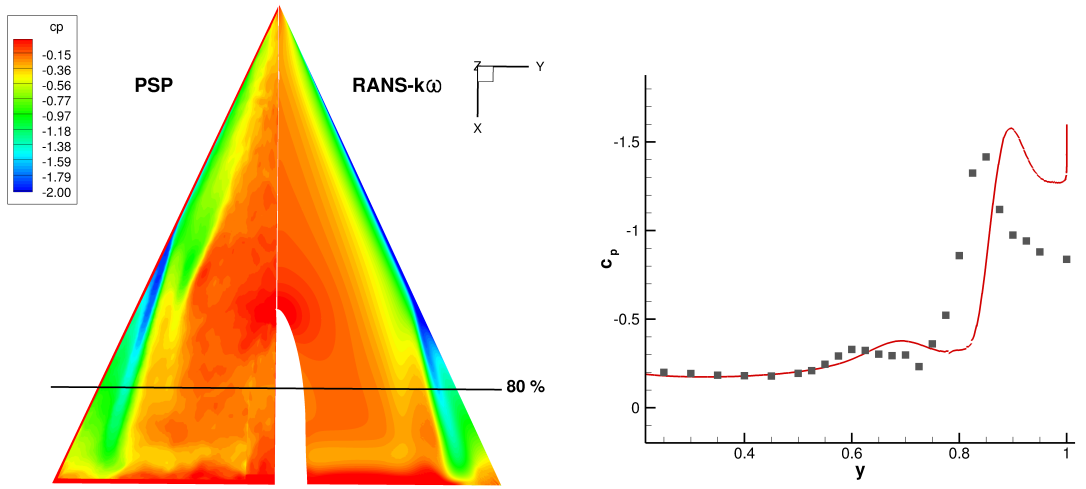

Figure 5.14: Comparison of c_p distribution for a $p = 2$ RANS-kw result on the mesh with 884 224 elements and experimental data. Left: On the upper wing with experimental data on the left and flow computation data on the right. Right: Slice at 80% chord of the upper wing.

Figure 5.14 shows the c_p distribution for a $p = 2$ RANS-kw result on the mesh with 884 224 elements compared to experimental data. For the comparison of the surface c_p distribution on the upper wing the numerical result shows the separation point of the main vortex further downstream as seen in Figure 5.14 on the left. The overall agreement of the numerical surface c_p distribution to the experimental data distribution seems to be in the range of other numerical results as seen in [2, 35]. The c_p distribution in a slice at 80% chord of the main upper wing is shown in Figure 5.14 on the right. The main characteristic of the experimental data, represented with ■, is also visible in the numerical data which is represented by a red line. A displacement of the numerical results of 0.05 in

y direction is explained with the later separation point of the vortices, as seen in the plot on the left.

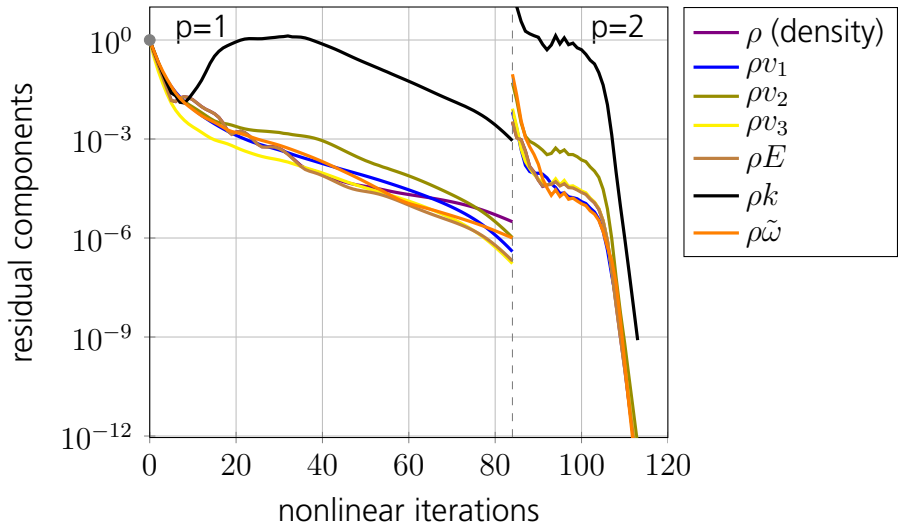

Figure 5.15: Nonlinear convergence of all residual components for an VFE-2 $k\omega$-computation at $p = 2$ on the 110 528 element mesh.

Figure 5.15 shows all residual components of a $p = 2$ RANS-$k\omega$ computation on the 110 528 element mesh. Here, the combination of a two level linear and nonlinear p-multigrid algorithm with a p-startup strategy is applied. Note that, even if only the density residual component is shown for this test case in a later section all other residual components then converge with the same rate and accuracy as the density residual component shown.

5.1.4 NASA Trap Wing

This test case is based on the geometry and flow conditions defined as "Configuration 1" of the First AIAA CFD High Lift Prediction Workshop, which is also called NASA Trap Wing. Detailed information and documentation can be found in [61]. The flow is treated as symmetric, thus a half model is considered. The flow conditions which are investigated here are defined by:

- Mach number M $= 0.2$,
- Reynolds number Re $= 4.3 \cdot 10^6$,
- angle of attack $\alpha = 13°$,
- side slip angle $\beta = 0°$.

These flow conditions represent the linear regime of the lift polar. Furthermore, for this test case higher order results from the EC funded IDIHOM project are available as well as experimental data from [61].

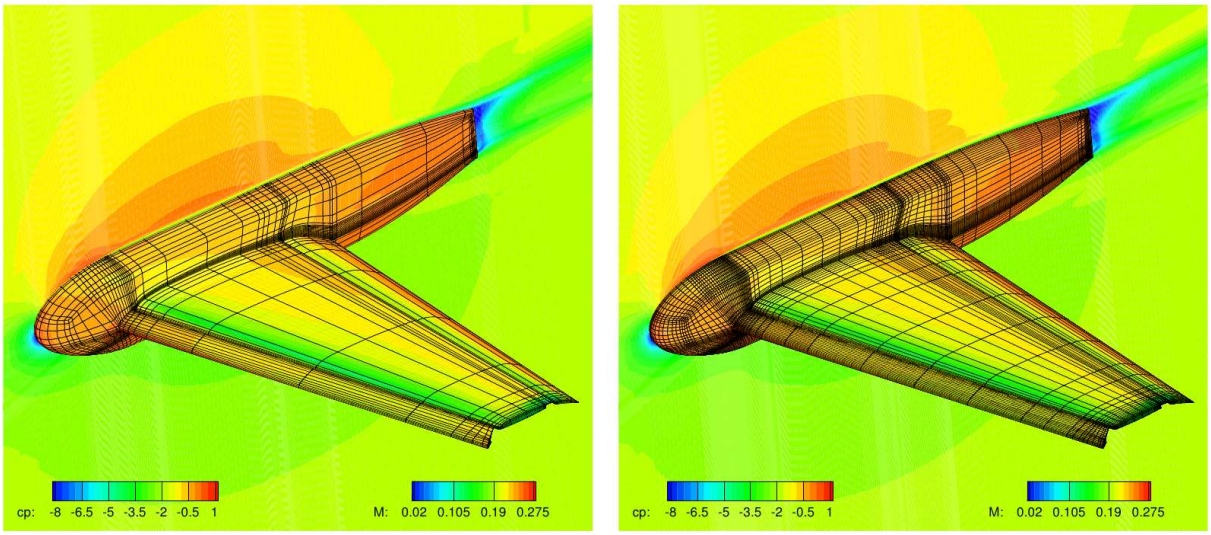

Figure 5.16: Mach number distribution plotted on the symmetry plane and the c_p distribution on the wall. Left: $p = 3$ SA-results on the coarse mesh with 93 088 elements. Right: $p = 2$ SA-results on the fine mesh with 744 704 elements.

Computations are shown on a hierarchy of two higher order hexahedral meshes with 93 088 and 744 704 elements, see Figure 5.16. For both meshes a quadratic mapping for the mesh curvature is applied. The wing is a high lift three element wing with the slat deflected by $30°$ and the flap deflected by $25°$. Moreover, the airfoil has a shroud with a $90°$ plane angle on the main wing, as seen for the MDA 30P30N test case.

Both plots shown in Figure 5.16 represent typical DG results for this configuration as seen in [35].

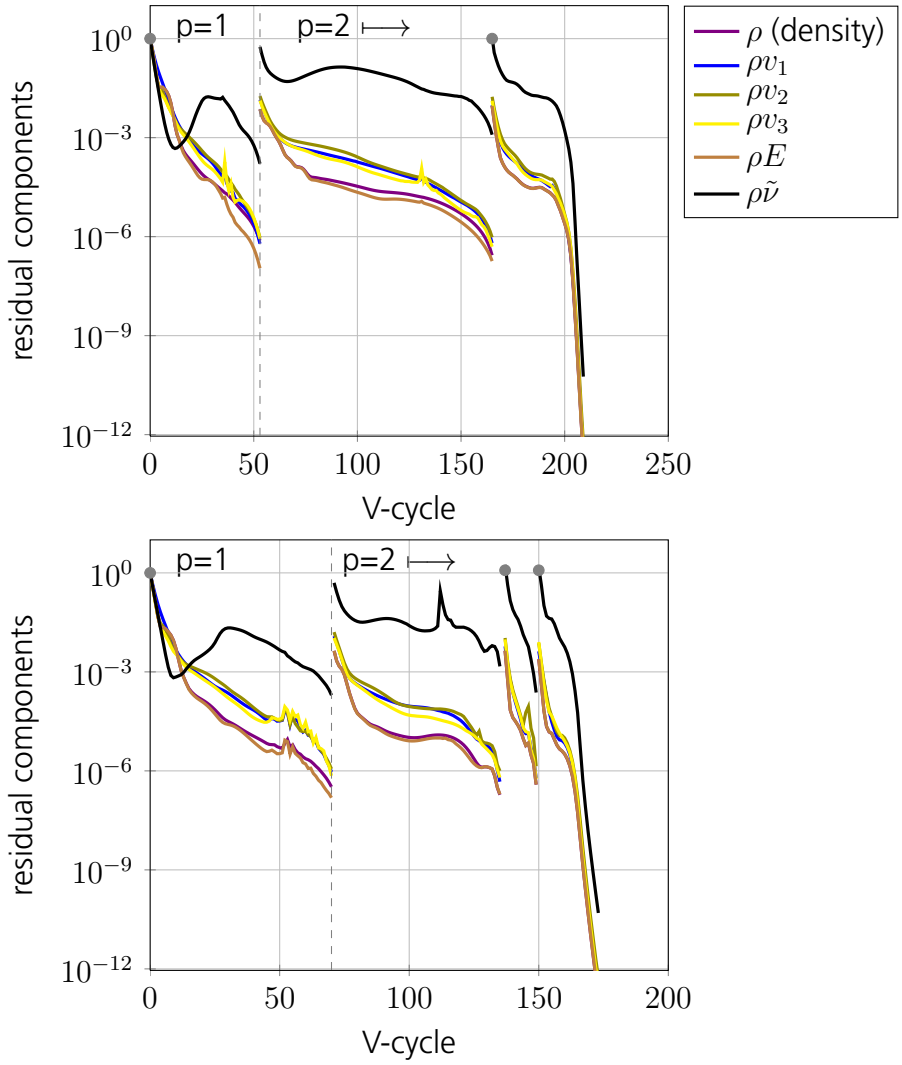

Figure 5.17: Nonlinear convergence of all residual components for the NASA Trap Wing RANS-SA computations for $p = 2$ on the 93 088 element mesh (top) and on the fine 744 704 element mesh (bottom).

In Figure 5.17 the convergence rate of two $p = 2$ RANS-SA computations on the 93 088 element mesh and on the fine 744 704 element mesh is shown. For both computations all residual components converge with the same rate. For the computation on the 93 088 element mesh a two level nonlinear h-multigrid algorithm with a Backward-Euler smoother and a linear h-multigrid preconditioned GMRes method to solve the linear problems is employed, whereas for the mesh with 744 704 elements three level multigrid algorithms are used. In the bottom plot of Figure 5.17 every equation is discretized with 7 447 040 DoF. Results on the coarse mesh have been computed for $p = 1, 2, 3$ and for the fine mesh results for $p = 1, 2$ are available. Higher order DG data computed on the same meshes and for the same polynomial degrees with the RANS-$k\omega$ equations is available from the University of Brescia and their CFD code Migale, which was also used by the University of Bergamo in the previous sections. These results are extracted from the IDIHOM project [33].

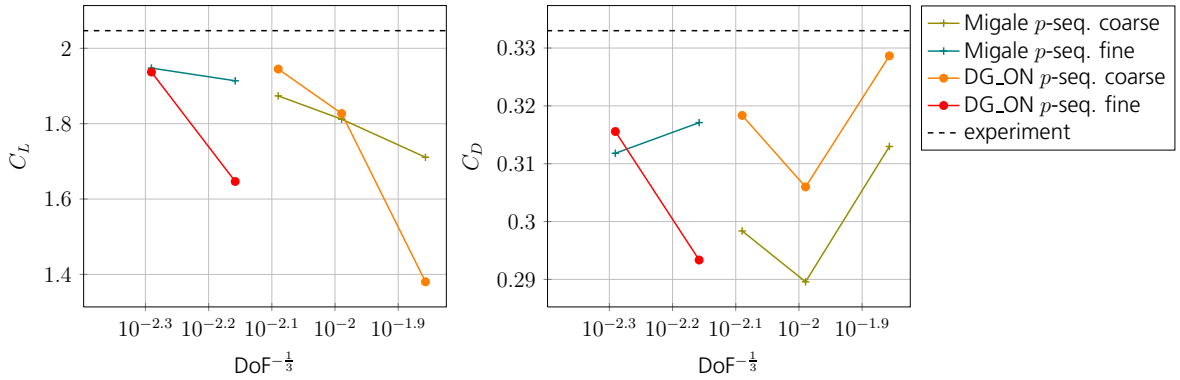

Figure 5.18: Convergence of lift (left) and drag (right) coefficient for high order computations in comparison to experimental data.

A comparison with this data in terms of the lift and drag coefficients is shown in Figure 5.18. For the lift coefficient the $p = 1$ results of the University of Brescia are closer to the experimental data than the DG_ON results, whereas the $p = 2$ results are very similar for both CFD codes. The RANS-SA results from DG_ON seem to show a better gain in accuracy by increasing the polynomial degree on the same mesh than the RANS-$k\omega$ results from the Migale code. For both codes the drag coefficient of the $p = 1$ solutions on the coarse mesh are very good in comparison to the experimental data. But these good results seem to be due to a beneficial integral error cancellation, since if the order is increased on the coarse mesh the $p = 2$ results differ much more from the experimental data whereas the $p = 3$ results start to move in the right direction again. Both codes show the same behavior on the coarse mesh, whereas the DG_ON results are closer to the experimental data, due to the difference in the applied turbulence models. On the fine mesh the Migale code shows similar results as shown for the coarse mesh. DG_ON however, shows the same behavior as seen for the lift coefficient on the fine mesh.

Finally, Figure 5.19 shows c_p distributions for different slices on the wing. The position where the slice is taken is specified by η, where the wing length is normalized to 1 and the body wing intersection is at $\eta = 0$.

The DG_ON results at 28% and 50% of the wing span show a good agreement with the experimental data. The $p = 3$ results on the coarse mesh tend to be a little smoother than the $p = 2$ results on the fine mesh in most regions of these slices overall. However, at the end of the slat the $p = 3$ tend to have a larger peak in the c_p distribution. The results shown at 95% of the wing span show a discrepancy with the experimental data at trailing edge of the main wing and the leading edge of the flap. However, these SA-results show a similar behavior as other SA results taken at this wing position from other CFD codes and therefore show a good agreement in this respect. Hence, this difference at the 95% of the wing span could be due to the turbulence model as seen by other authors in [35].

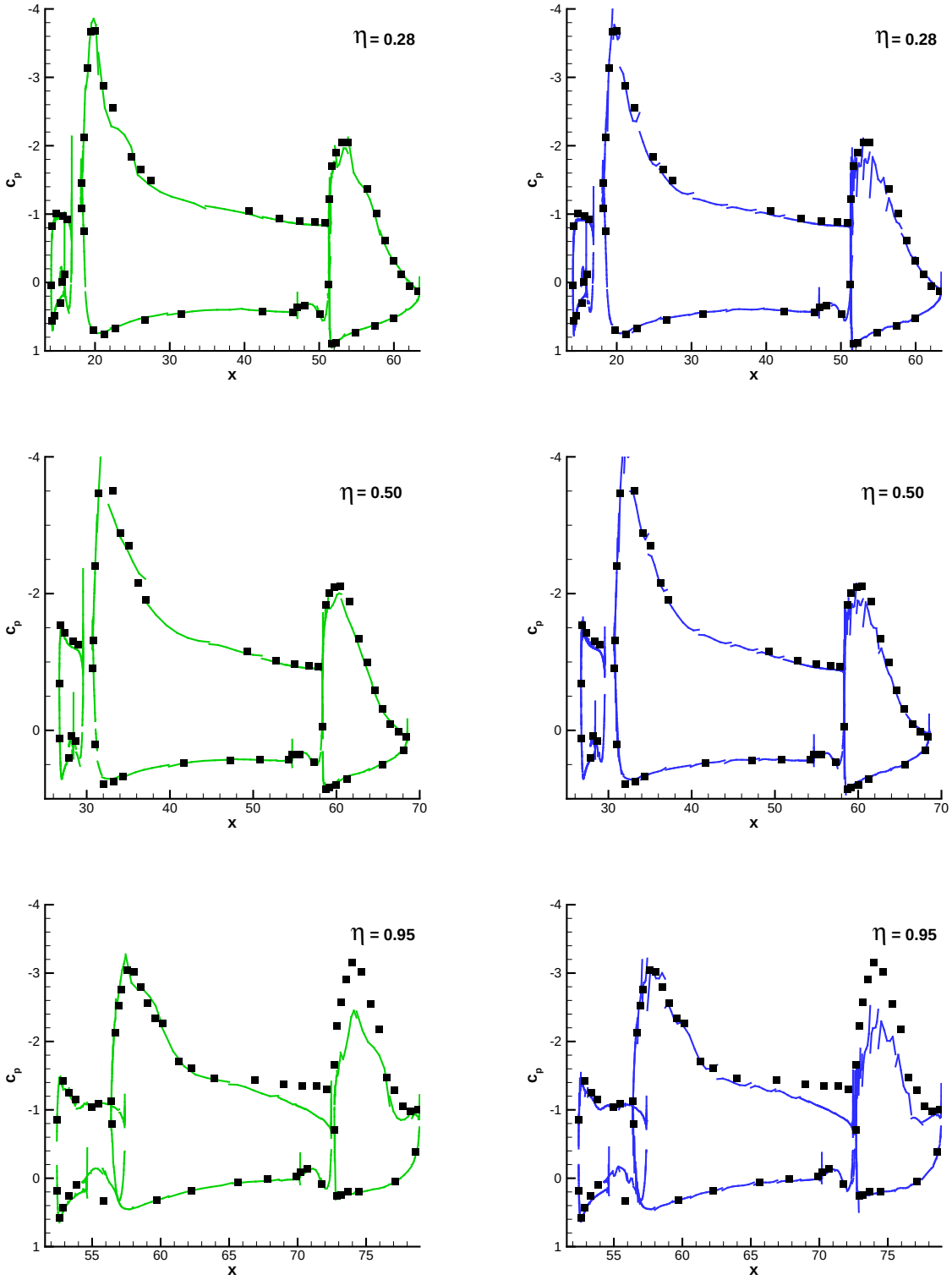

Figure 5.19: Comparison of the c_p distribution at different slices on the wing for $p = 3$ SA-results on the coarse mesh with 93 088 elements (left), $p = 2$ SA-results on the fine mesh with 744 704 elements (right) and experimental data ■.

5.2 Algorithmic investigations

In this section, investigations are performed within the DG_ON framework, among others comparisons of the proposed multigrid algorithms with single-level solver in combination with the most common solver strategies in DG methods. The fact that all solver approaches discussed in Section 5.2 are implemented in the same framework, and therefore use the same components, e. g. like timestep control, discretization, Jacobian, residual evaluation, etc., allows to draw comprehensive conclusions. Hence, if a similar single-level solver is already implemented in a CFD code, it can be expected to achieve similar improvements by implementing the "best practice" solver shown in this work. As a brief perspective of what the "best practice" solver within DG_ON is capable off, the following numerical example is given. Most CFD codes using DG methods, e. g. the DLR PADGE code [27], do not have any agglomeration options. Hence, if a DG result for a given polynomial degree p is desired on an unstructured mesh only two options remain.

The first option is to start the computation on the given mesh for the desired polynomial degree, see the unmarked lines in Figure 5.20, the second option is to perform a p-startup strategy, see the lines market with a "$+$" in Figure 5.20. For both options, a single-level Backward-Euler **Algorithm 2** (BWE) with a Line–Jacobi **Algorithm 4** preconditioned GMRes method is used. Both options can be considered standard approaches and the state of the art solver for turbulent computations on unstructured meshes in the DG community [5, 14, 26, 76]. In detail, Figure 5.20 shows L1T2 test case computations of the RANS-$k\omega$ equations for a $p = 2$ DG discretization on the CEN mesh with 23 824 elements, see Section 5.1.1. Note that for each computation shown in Figure 5.20 the solver settings, e.g. initial CFL number and max. GMRes steps, are applied which converge the solution in the fastest possible CPU time. The lines marked with a "$\bullet$" in Figure 5.20 represent the "best practice" approach within DG_ON. Here, a combination of the nonlinear and linear h-multigrid algorithm is employed with a p-startup strategy on the coarsest agglomerate in combination with a h-startup strategy using all agglomerated meshes involved.

As can be seen in Figure 5.20, this approach is superior in terms of number of nonlinear iterations and overall CPU time required to converge to the same numerical solution like the other computations. For the multigrid computation only the smoother iterations on the current top level of the startup strategy are plotted in Figure 5.20 (top). Note that each smoother step on the top level corresponds to one iteration of the BWE **Algorithm 2**. For single-grid BWE without startup strategy, represented by the unmarked line in Figure 5.20, 806 BWE iterations are required on the CEN mesh to obtain a converged solution. By applying the same algorithms in combination with a p-startup strategy only 192 BWE $p = 2$ iterations are necessary. The "best practice" approach only needs 96

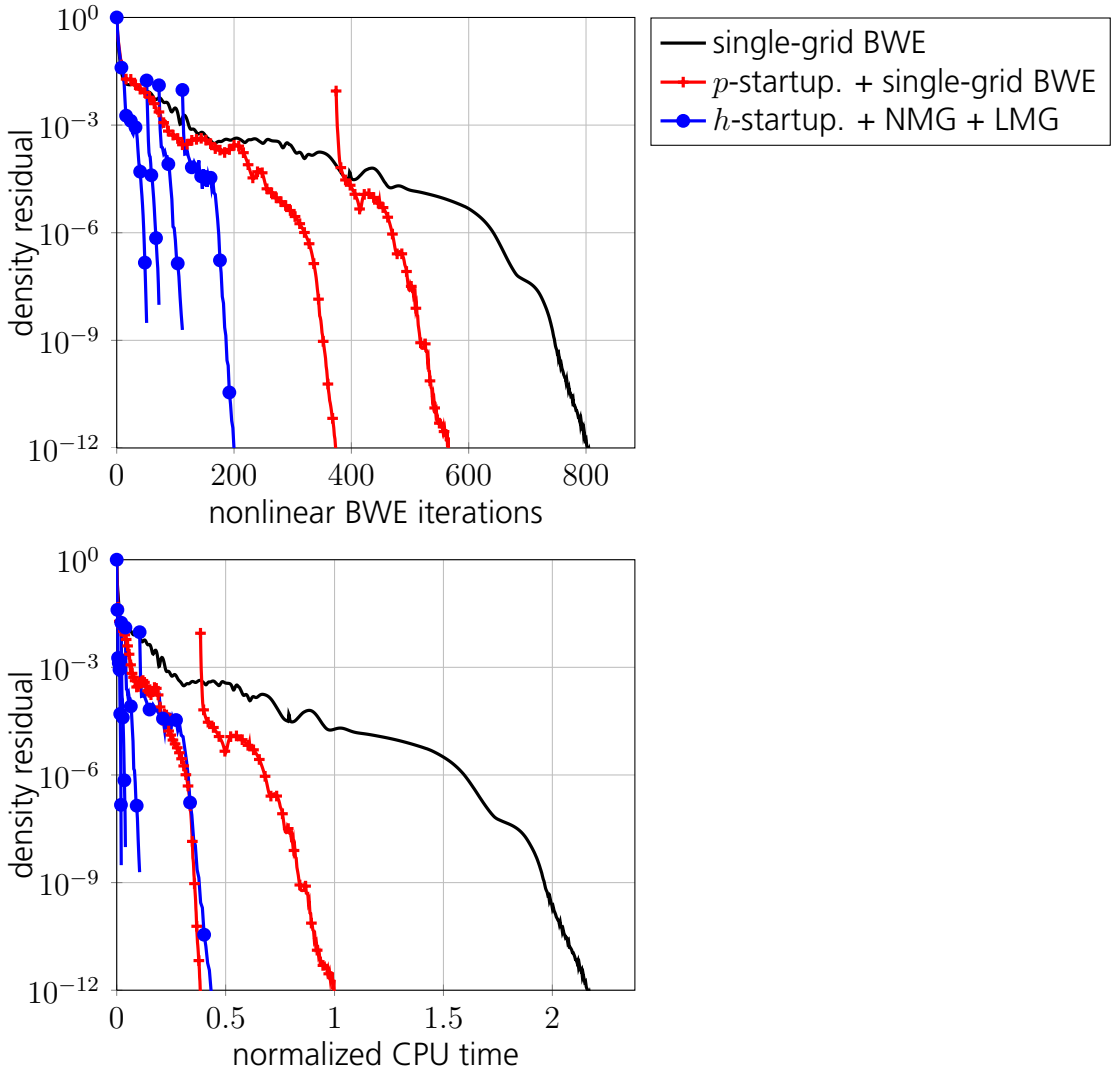

Figure 5.20: Nonlinear convergence for a L1T2 $k\omega$-computation at $p = 2$ on the CEN mesh for different combinations of solver algorithms.

smoother steps, and therefore BWE iterations, on the top level of the h-startup strategy and h-multigrid algorithm. Hence, the number of nonlinear BWE iterations on the fine mesh for $p = 2$, which have the highest cost in terms of CPU time and memory, could be reduced by a factor of 2 for the "best practice" approach in comparison to the p-startup strategy single-grid computation. Furthermore, a similar factor can be observed in terms of overall CPU time between both computations, see Figure 5.20 (bottom). Even larger factors can be observed when no p-startup strategy is applied, see the unmarked lines in Figure 5.20. This is a typical result for this framework. In fact, similar results have been obtained on every mesh and for every applied polynomial degree considered in this work. While the exact factor of reduction in CPU time may vary a superior CPU time can always be observed.

In the following, single components and their behavior within the DG_ON framework will be investigated as to how they contribute to the improvement seen in Figure 5.20

in comparison to the single-grid computations. Before presenting the investigations of the proposed framework, the structure of the tables used to summarize and compare the results are briefly explained and some general settings are introduced in in Section 5.2.1.

5.2.1 General settings and information

This section describes the column headers of the tables which will occur in the following sections. Not every column mentioned in this section will necessarily be included in each table. The focus on which columns are given depend on the particular aspect of the investigation under consideration. If a column is not included in a table it means that the data in this column would be identical for all computations presented in the table and have no importance for the investigations under consideration. In the following each column is described:

- "no.":
  This column is included in every table and indicates the row number.

- "startup":
  How startup strategies work and which possibilities exist have been explained in Section 4.5. The entries of this column indicate how many orders of magnitude the nonlinear residual is reduced on every level of the startup strategy. In particular, this column specifies the residual tolerance r_{tol} of **Algorithm 5**. An entry like "-" in this particular column means that no startup strategy is applied and the computations start for a desired polynomial degree p_{max} on a given mesh. Moreover, an entry "1" in this column means that a one-step startup strategy is employed, which performs one nonlinear iteration on every startup level and uses this result as initial solution for the next startup strategy level.

- "levels":
  This column indicates how many levels are used and whether it is a p- or h-startup strategy or multigrid algorithm. A p-startup with p_{max} levels is indicated by $p(p_{\text{max}})$ in this column. The second option, $h(n)$, with $n \in \mathbb{N}$, indicates an h-startup strategy, where n determines how many meshes are used during the computation including both agglomerated meshes and the underlying fine mesh. Recall that both startup strategies have been introduced in Section 4.5. Also recall that order sequencing can be considered a standard technique for high order finite element (FE) methods in nonlinear applications. Note that this column also specifies how many levels are used on the top level of the startup strategy if multigrid algorithms are used. In particular, the multigrid algorithms use the same level sequence as the startup strategy.

- "NMG":
  This column refers to the cycle type within a nonlinear multigrid, if any in use. An entry "-" means that only a single-level Backward–Euler algorithm is employed. Furthermore, possible entry choices are a V-cycle "v" and a W-cycle "w". Moreover, the type of the executed nonlinear multigrid is also specified by the column "levels". Note, that a "-" entry in this column and a $p(p_{\max})$ or $h(n)$ in the column "levels" is no contradiction. It just indicates that the lower levels of the startup strategy are not used to solve the nonlinear problems on the higher levels, i.e. no nonlinear multigrid algorithm is used. Moreover, for all numerical examples using the nonlinear multigrid algorithm m_1 is equal to m_2 in **Algorithm 1**, which means that the same number of pre- and post-smoothing steps are performed on each level. However, the number of smoothing steps can differ between levels. In that case, for each level m_1 is given. An entry in this column can therefore look like this "v$(10\,5\,5)$" which means a 3 level V-cycle is applied and the nonlinear smoother steps are specified beginning with the lowest level. In the case the column "NMG" only contains the cycle type then m_1 is equal on all levels, even on the lowest level.

- "LMG":
  This column describes whether or not a linear multigrid is used within a computation. When the linear multigrid is used the cycle type appears in the column. The options for these cycle types and smoother steps are the same as for the column "NMG", plus an additional cycle type, namely the sawtooth-cycle "/". For the sawtooth-cycle m_1 from **Algorithm 3** is always equal to zero and only post-smoothing steps are performed which will be specified like described for the "NMG" column.

- "coarse lin.":
  In case a multigrid algorithm is used during a computation this column indicates how the Jacobian matrix within the Backward–Euler iteration is computed on the lower levels. The first option is a straight-forward block-wise reassembling of the Jacobian using the restriction of the solution from a higher level, which is denoted as "redisc.", short for rediscretization. The second option is a transfer of the fine level Jacobian via a Galerkin transfer as seen in Section 4.6, which is indicated with "GT". This column is mainly used for the investigations in Section 5.2.4.

- "LMG it.", "GMRes it.":
  These columns include the maximal number of iterations for both algorithms, the linear multigrid algorithm and the GMRes method, respectively. If there is one number given only then the maximal number of iterations is the same on all levels of the computation. Otherwise these columns include the maximal number of iterations

per multigrid algorithm or startup strategy level, starting with the number for the lowest level. Note that these maximal numbers are not always reached in the solution process. In fact, if the linear residual is reduced six orders of magnitude relative to the start residual, the linear solving process will also stop. When none of these columns is given, it could be that no maximal iteration number was provided and the reduction threshold was set to a lower value or kept at 10^{-6}. If this is the case it will be mentioned in the explanatory text of the table.

- "ave. LMG it.", "ave. GMRes it.":
  These columns include the average number of steps performed with these algorithms. Here every linear iteration on every level of the startup strategy is counted and divided by the number of nonlinear iterations on every level. In the case of a nonlinear multigrid algorithm used these columns are not given.

- "CPU time":
  This column includes the time of the computation which is normalized with a case-dependent reference time. For each mesh and discretization in combination with a certain number of processors the CPU time of one computation for this case is chosen and used as the normalizing factor for all the other computations on this specific mesh with the same polynomial degree and number of processors used. In most cases the computational time for a single-level computation in combination with an h-startup strategy is chosen as the reference. Moreover, this column reports the CPU time of the whole computation including the computational time spend on startup strategies.

- "nonlinear iter.":
  This column includes the number of Backward–Euler iterations on every level of the startup strategy. When a nonlinear multigrid algorithm from Section 4.1 is chosen the smoother is the Backward–Euler algorithm. For comparability with single-level and linear multigrid computations the number of current fine level smoother steps are given, neglecting the lower level smoother steps perfromed in the nonlinear multigrid cycle. Moreover, if an h-startup strategy is used, this column includes the number of nonlinear iterations on the occurring top levels of the startup strategy for p_{max} only, neglecting the p-startup strategy on the coarsest agglomerate.

- "nonlinear cycles":
  This column includes the number nonlinear multigrid cycles on every level of the startup strategy. One the coarsest level of the startup strategy the entry is identical to a single-grid solver. For comparability between nonlinear multigrid computations this column is used and never stated if a single grid computation is also shown in

the same table. Moreover, if an h-startup strategy is used, this column includes the number of nonlinear cycles on the occurring top levels of the startup strategy for p_{max} only, neglecting the p-startup strategy on the coarsest agglomerate.

In the case of an h-startup strategy, the p-startup strategy on the coarsest agglomerate is performed in a single level mode, i. e. no p-multigrid is applied on agglomerated meshes. In order to handle a combination of p- and h-multigrid algorithms within the proposed framework more implementational work has to be performed and therefore, a investigation of these variations is beyond the scope of the current work.

All structured meshes used in this work have been generated based on fine block-structured FV meshes, which are coarsened uniformly, see Section 3.5.

A computation in this work is classified as converged if the nonlinear residual components on a given mesh for a desired polynomial degree are reduced to 10^{-12}, except of the k- and $\tilde{\nu}$-components, after normalization with the associated freestream residual components. As seen in Section 5.1, the residual components for the k- and $\tilde{\nu}$-entries converge with the same rate but they are usually three order of magnitude larger than all other components after normalization.

5.2.2 Investigation of startup strategies

In this section the different startup strategies introduced in Section 4.5 are investigated. The influence of the residual tolerance r_{tol} of the coarser level will be the main focus after the general need of a startup strategy for RANS computations is shown. This residual tolerance specifies how much a solution has to be converged on the lower level in order to be usable as initial solution for the next level of the startup strategy. Moreover, for both investigated turbulence models possible distinctions in the startup strategy are investigated.

Necessity of startup strategies

First of all a computation with startup strategy can always outperform a computation without a startup strategy in terms of the number of top level iterations and CPU time. This is due to the fact that an initial solution on the top level which is "better" than freestream values makes the overall computation more robust and allows larger initial CFL numbers.

Since one of the main side conditions in this and all following sections is comparability, the same SER time step control (4.3) and CFL number is used at every point of the nonlinear convergence history for each computation in a single figure or table. This means that if the nonlinear residual is the same for two computations they both employ the same CFL number CFL_i. Recall that in the SER timestep control the CFL number CFL_i at a certain iteration in the convergence history depends on the initial CFL number CFL_0 and the residual reduction from freestream.

A startup strategy already provides an initial residual reduction on the top level but in comparison with a computation started from freestream the same timestep control and CFL number will be employed. This means that if both computations have the same nonlinear residual the same CFL number is employed, for comparability reasons.

However, due to the startup strategy a larger CFL number could be chosen at the level starting point and the associated nonlinear residual instead of the CFL number originating from a freestream start and altered by the SER timestep control at the same residual. This is due to the fact that, the critical part of all computations are the first iterations where the initial solution differs most from the converged solution. This implies that the SER time step control does not choose optimal CFL numbers for every polynomial reduction. If with the help of a startup strategy the starting point on the top level is moved to an smaller initial residual a larger CFL number could be chosen at this point.

To converge a RANS-$k\omega$ solution on a given mesh for a given polynomial degree without any kind of startup strategy is very difficult and it gets even more challenging if the meshes get finer or the polynomial degree is increased. For most test cases, if possible at all, it is very hard and time consuming to create such results, e.g. for the MDA 30P30N testcase

on the 33 728 element mesh it was not possible to even compute a $p = 2$ solution without a startup strategy, whereas using a startup strategy a solution could be easily obtained. On the same mesh with the same polynomial degree there is no problem to compute RANS-SA solutions without a startup strategy. This observation is typical for the RANS-SA discretization in comparison to the RANS-$k\omega$ discretization. In fact, it is significantly easier to compute RANS-SA solutions than RANS-$k\omega$ solutions on a given mesh for a given polynomial degree. Moreover, if a solution can be computed for both turbulence models on a given mesh without a startup strategy then the RANS-SA computation is much faster and takes less iterations for comparable solver settings. Nevertheless a startup strategy is essential to enhance the solution procedure for both turbulence models.

In Figure 5.21 a mesh and polynomial degree was chosen where a solution can be computed for both turbulence models without a startup strategy. The CPU time is normalized with the computational time of the RANS-SA computation and the same number of GMRes steps were applied. For the RANS-SA computation the CFL number could be chosen larger and the timestep control stronger resulting in a faster convergence. Solving the linear problems with a Line–Jacobi preconditioned GMRes method the RANS-$k\omega$ computation takes 809 Backward–Euler iterations, whereas the RANS-SA computation takes 431 iterations only. This is a typical result for these kind of computations.

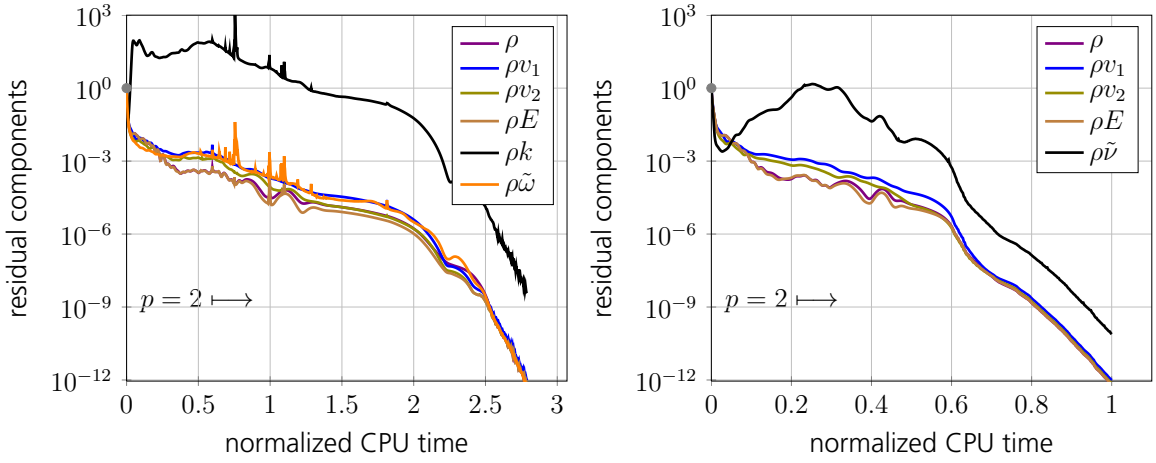

Figure 5.21: Nonlinear convergence of all residual components for $p = 2$ computation of the L1T2 on the CEN mesh for the RANS-$k\omega$ equations (left) and the RANS-SA equations (right) with no startup strategy applied.

In Figure 5.21 the l^2-norms of all residual components are shown for both turbulence models for a $p = 2$ computation on the same mesh and without a startup strategy employed. Here a typical behavior of the RANS-$k\omega$ equations can be observed. When the l^2-norms of the residual components are normalized with the l^2-norms of the freestream residual components the k-residual component tends to fall like all other residual components but then rises again although all other residual components are reduced. After a certain time the k-residual component starts to reduce again with the same rate as

all other components. This behavior can be observed for every DG_ON computation for the RANS-$k\omega$ equations, see Section 5.1 for more examples. A comparable behavior can be observed for the RANS-SA equations and the $\tilde{\nu}$-residual component. Hence, this freestream flow field adjustment to a airfoil in the flow field in the early stages of the iteration process is observed for both turbulence models and is most visible in the k- or $\tilde{\nu}$-residual component. The main reason why startup strategies are beneficial is that this freestream flow field adjustment takes place on lower levels and on the higher levels only local changes are required on the higher levels. In fact, startup strategies can help to reach this adjustment faster or remove the need for the adjustment on the top level of the startup strategy at all, see Figure 5.22.

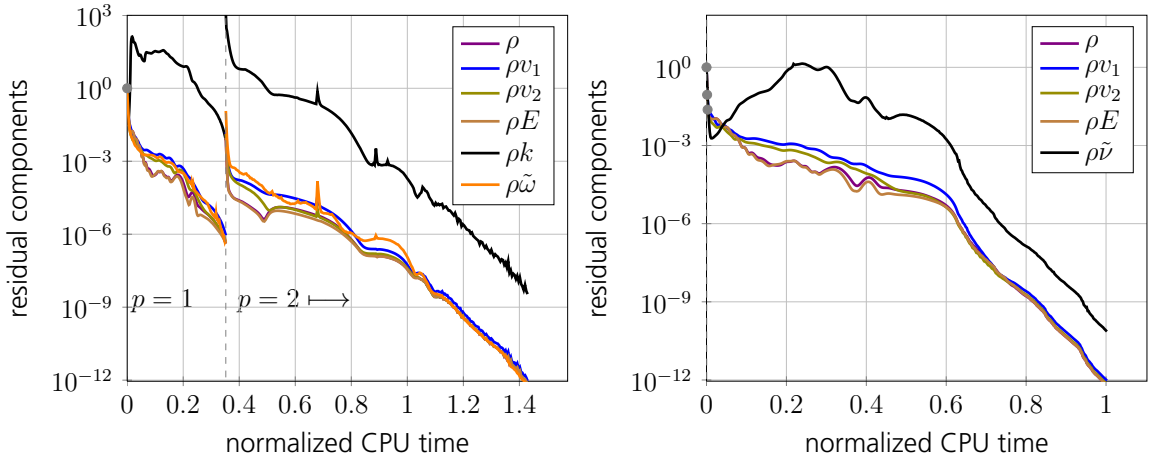

Figure 5.22: Nonlinear convergence of all residual components for $p = 2$ computation of the L1T2 on the CEN mesh for the RANS-$k\omega$ equations (left) and the RANS-SA equations (right) with startup strategy applied.

In Figure 5.22 two different startup strategies are employed. In the RANS-$k\omega$ case, see Figure 5.22 on the left, a p-startup strategy is applied with a residual tolerance of 10^{-6}. This reduces the normalized CPU time from 2.79 without a startup strategy to 1.43. The same timestep control as in Figure 5.21 is applied in this figure. Hence, the same CFL number is used at the same nonlinear residual on the top level of the startup strategy like in the computation without a startup strategy. The reduction in CPU time could be even larger if a higher initial CFL number is chosen for the startup strategy computation, for which the computation without startup strategy would encounter instabilities. Following from this, the overall CPU time is reduced because the initial freestream flow adjusting to the airfoil in the domain on the lower levels on the startup strategy where the k-component of the residual increases and after a while converges with the same rate as all other components. Recall that, the iteration cost on lower levels is very cheap in comparison to the $p = 2$ iterations. Then the flow on the $p = 2$ level is already adjusted to the flow problem and will converge much faster with only 237 nonlinear iterations

because only local flow changes are required. This is a typical result for all RANS-$k\omega$ computations.

In the RANS-SA case, see Figure 5.22 on the right, a one step 3 level h-startup strategy is applied. Please recall that in every h-startup strategy a p-startup strategy is performed on the coarsest grid of the h-startup strategy. This is due to the fact, that a p-startup strategy is always beneficial in terms of robustness and most of the time also in terms of CPU time. The CPU time is normalized like in the RANS-SA computation without a startup strategy in Figure 5.21, but this startup strategy computation takes 13 iterations less on the top level. However, this reduction in nonlinear iterations is not seen in CPU time since the startup strategy seems to spend exactly the saved CPU time from this iteration reduction. Here the same timestep control is employed as in Figure 5.21 (right). A reduction in CPU time can be achieved if the initial CFL number of the startup computation is chosen a little larger where the other computation would encounter instabilities. In order to see similar reductions in CPU time and number of nonlinear iterations finer meshes or higher polynomial degrees need to be investigated. Hence, for the RANS-SA equations an approach different from a one step startup strategy is required only if fine meshes or higher polynomial degrees are considered.

Difference of p- and h-startup strategies

In the following the p- and h-startup strategies will be compared with respect to the number of level iterations and CPU time. In a p-startup strategy the initial solution on each level is taken from a computation on the same mesh but from a lower polynomial degree. Furthermore, in an h-startup strategy the initial solution on a mesh is taken from a computation on agglomerated mesh but with the same polynomial degree.

Table 5.5 shows in row 1 the p-startup strategy applied to the L1T2 test case on the CEN mesh. When these results are compared to the p-startup computation shown in Figure 5.22 for the RANS-$k\omega$ equations about half the iterations on the top level of the startup strategy are used. This is due to the fact that for the computation in row 1 of Table 5.5 a stronger timestep control and a larger initial CFL number is chosen. The computation in Figure 5.22 used such a low initial CFL number because of the comparability with the computation without a startup strategy. The computations with a h-startup strategy presented in the Tables 5.5 - 5.7 are superior in terms of CPU time in comparison to the p-startup strategy computations. Most notably is that for the MDA 30P30N test case in Table 5.6 even for the same time step control on the top level 54 nonlinear Backward–Euler iterations are saved, although exactly the same single-grid solver is applied. A similar result is obtained for the RANS-SA computation of the NASA Trap Wing test case in Table 5.7. The main reduction in the number of nonlinear iterations on the top level of the startup strategies is due to the fact, that the initial residual on the top level is usually much smaller when an h-startup strategy is performed in comparison to a p-startup strategy. This

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-8}	$p(2)$	1.00	279, 218
2	10^{-8}	$h(3)$	0.87	21, 91, 220

Table 5.5: L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: p- and h-startup strategy investigation.

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-6}	$p(2)$	1.00	132, 139
2	10^{-6}	$h(2)$	0.74	40, 85

Table 5.6: MDA 30P30N $(p = 2)$ $k\omega$-computation on the 8 432 element mesh: p- and h-startup strategy investigation.

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-6}	$p(2)$	1.00	162, 321
2	10^{-6}	$h(3)$	0.75	64, 39, 182

Table 5.7: NASA Trap Wing $(p = 2)$ SA-computation on the 744 704 element mesh: p- and h-startup strategy investigation.

indicates that an h-startup strategy which enhances the top level with a initial solution originating from the same polynomial degree but from a coarser mesh is better than a p-startup strategy where the initial solution originates from the same mesh but for a lower polynomial degree. Note that, these results are common for these kinds of startup strategies if within the h-startup strategy a p-startup strategy is performed on the coarsest mesh. However, some meshes are too coarse or have a bad agglomeration behavior. For these cases a p-startup strategy still can be a useful approach. Moreover, an h-startup strategy is also more robust than the p-counterpart.

Residual tolerance and one step startup strategy

The coarse level residual tolerance r_{tol} plays an important role with respect to the efficiency of the start-up procedure. A simple one step startup strategy will be investigated along side. The one step startup strategy can be triggered by setting b_{it}=false in **Algorithm 5** and will perform one iteration on every level of the startup strategy in contrast to reducing the nonlinear residual to a certain residual tolerance on each level.

Figure 5.23 shows $p = 3$ computations of the L1T2 test case for the RANS-SA equations on the WUT mesh for the different 3 level h-startup strategies. The black unmarked line represents a one step startup strategy which performs one $p = 1$ iteration on the coarsest grid, uses this as an initial solution for one $p = 2$ iterations on the coarsest grid, which is then used as an initial solution for one $p = 3$ iteration on the coarsest grid. After that one $p = 3$ iteration on the next agglomerate is performed and then used as an initial solution on the original WUT mesh for a $p = 3$ computation. The blue line marked with a

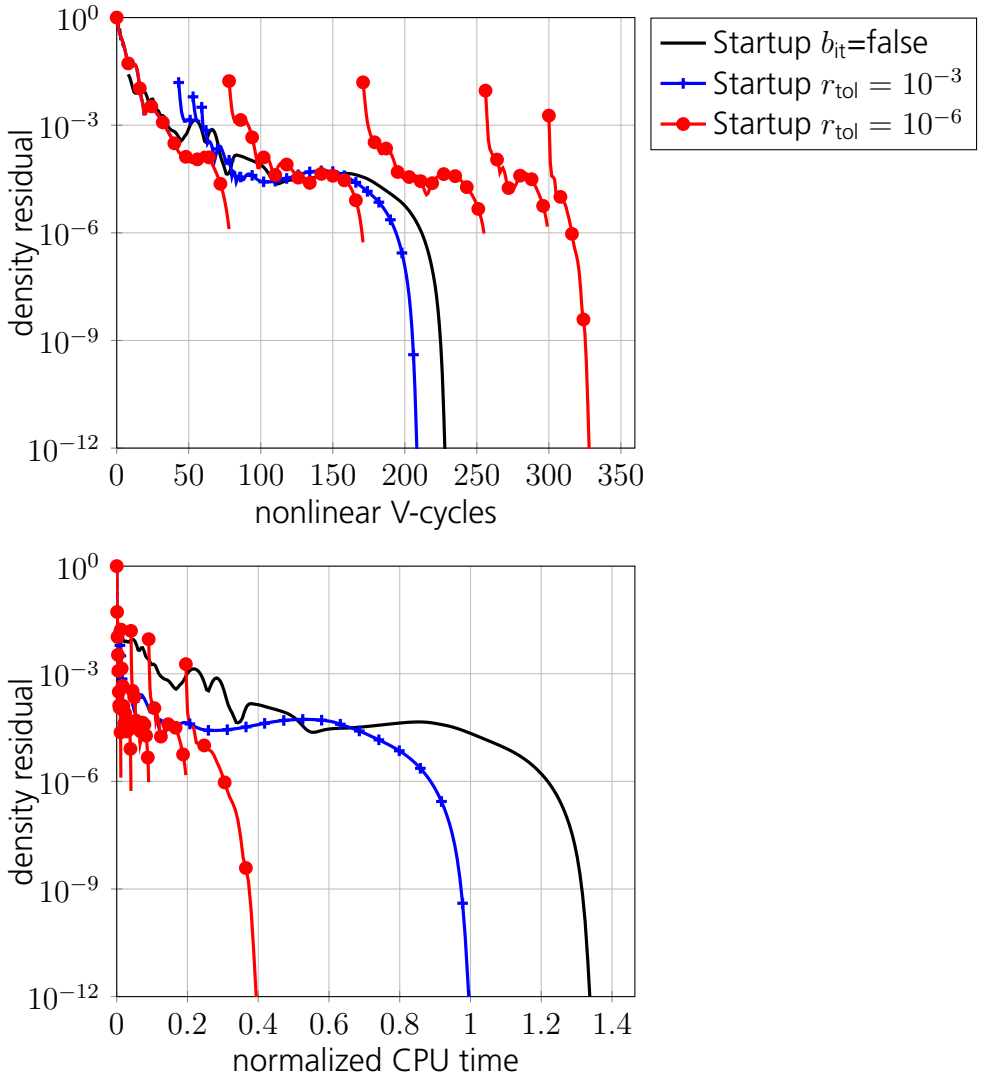

Figure 5.23: Nonlinear convergence for a L1T2 SA-computation at $p = 3$ on the WUT mesh for different h-startup strategy tolerances.

"$+$" represents the same startup strategy as described before but instead of performing one nonlinear iteration the nonlinear residual is reduced to 10^{-3} on each level of the startup strategy. The red line marked with a "$\bullet$" uses a residual tolerance of 10^{-6}. For all computations the same timestep control and initial CFL numbers on every level are used. In terms of the number of nonlinear iterations on the top level the one step startup strategy requires 220 nonlinear V-cycles to converge whereas the computation with a residual tolerance of 10^{-3} requires 147 only. However, for both startup strategies the initial freestream flow adjusting to the airfoil is not completed on the lower levels of the startup strategy and, therefore, many nonlinear V-cycles are required on the top level of the startup strategy. The red line marked with a "$\bullet$" where the residual tolerance is set to 10^{-6} this freestream flow adjustment is already performed during the $p = 1$ computation on the coarsest grid. Hence, this computation requires 28 nonlinear V-cycles only on the top level of the startup strategy in order to converge. The same time step control is

applied and, therefore, the CFL number for a certain range of the nonlinear residual is the same for all computations. The initial residual in the v_1-component for the one step startup strategy on the top level is equal to 0.0261 if normalized with the appropriate freestream residual component. For the two other startup strategies this value is 0.0014 for the residual tolerance of 10^{-3} and 0.0025 for the residual tolerance of 10^{-6} startup strategy. Hence, the CFL number is smaller at the beginning of the one step startup strategy in comparison to the beginning of the other two startup strategies. When the residual of the v_1-component is equal to the initial values of the other startup strategies the CFL number has the same value as for the other computations at that point. So no gain in terms of timestep control can be expected. The only difference is that the solution process might look different but has the same residual range, e.g. for the computations with a residual tolerance of 10^{-3} and 10^{-6} the initial residual at the top level for $r_{\text{tol}} = 10^{-3}$ is a slightly smaller but the computation for $r_{\text{tol}} = 10^{-6}$ converges much faster on the top level. This is all due to the initial freestream flow field adjustment which was better on the lower levels of the startup strategy with $r_{\text{tol}} = 10^{-6}$. This result is typical for both turbulence models and startup strategies. Note that for the computation with $r_{\text{tol}} = 10^{-6}$ a higher CFL number could be used which converges even faster, the same time step control was chosen for comparability reasons, only.

For a residual tolerance of 10^{-12} the startup strategy uses on every level converged solutions from the next lower level. Hence, for a p-startup strategy converged $p = 1$ solution is used as initial solution for the $p = 2$ computation. Most codes without startup strategy implementation use this approach in order to obtain a good initial guess for the solution of the desired polynomial degree on the given mesh. A comparison of computations with a residual tolerance of 10^{-6} and 10^{-12} are shown in Table 5.8 - 5.11.

This investigation shows that a fully converged solution on a lower level as initial solution for the next level of the startup strategy is not essential to achieve the best reduction in the number of nonlinear iterations and CPU time on the top level of the startup strategy. Moreover, in all cases the computations with $r_{\text{tol}} = 10^{-12}$ requires more CPU time than the computations with $r_{\text{tol}} = 10^{-6}$. This is due to the fact that the nonlinear iterations on the top level stay the same or even decrease and the additional CPU time which is spent on reducing the residual on the lower levels to 10^{-12} increases the overall CPU time of the computation in comparison to a computation with $r_{\text{tol}} = 10^{-6}$.

Conclusion

For the proposed solver algorithms, every kind of startup strategy is beneficial in comparison to a computation without a startup strategy. The initial residual on the top level of a startup strategy is always smaller if an h-startup strategy is applied in comparison to a p-startup strategy with the same startup tolerance. Moreover, an h-startup strategy is

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-12}	$p(2)$	1.00	320, 449
2	10^{-6}	$p(2)$	0.96	270, 446

Table 5.8: L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: startup tolerance investigation.

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-12}	$p(2)$	1.00	124, 79
2	10^{-6}	$p(2)$	0.96	109, 79

Table 5.9: VFE-2 $(p = 2)$ $k\omega$-computation on a structured 13 816 element mesh: startup tolerance investigation.

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-12}	$h(3)$	1.00	105, 102, 58
2	10^{-6}	$h(3)$	0.89	84, 86, 56

Table 5.10: L1T2 $(p = 3)$ SA-computation on the WUT mesh: startup tolerance investigation.

no.	startup	levels	CPU time	nonlinear iter.
1	10^{-12}	$p(2)$	1.00	183, 104
2	10^{-6}	$p(2)$	0.93	145, 104

Table 5.11: NASA Trap Wing $(p = 2)$ SA-computation on the 744 704 element mesh: startup tolerance investigation.

more robust than the p-counterpart. However, which startup strategy is faster is clearly case dependent. For medium and fine meshes an h-startup strategy is favorable.

For the RANS-$k\omega$ equations it is beneficial to use a startup tolerance which allows the initial freestream flow to adjust to the airfoil on the lower levels of the startup strategy. This effect is most visible in the k-equation, since the k-equation residual tends to rise at the beginning of the solution process. When this freestream adjustment happens on the lowest level of the startup strategy, where the iterations are the cheapest, only local changes in the flowfield are necessary on the following levels. Thus, the number of iterations on the following levels are reduced. If the startup tolerance is chosen to high this freestream adjustment to the airfoil will happen on the top level of the startup strategy where the nonlinear iterations cost is the highest. The same observation can be drawn for the RANS-SA equations, in particular for the $\tilde{\nu}$-equation. However, as there the restart is not as important for coarse to medium sized meshes and a moderate polynomial degree most of the time a one step startup strategy for the RANS-SA equations is sufficient.

As a best practice the startup tolerance will be chosen as $r_{\text{tol}} = 10^{-6}$ for the RANS-$k\omega$ equations and for the RANS-SA equations on fine meshes or for higher polynomial

degrees. There is no need to choose a smaller tolerance since the iterations on the top level will stay the same in nearly all test cases. Furthermore, with a higher tolerance the initial flow field is possibly not sufficiently adjusted to the the airfoil on lower levels and more than local changes to the flowfield are required on the top level.

5.2.3 Trade-off between nonlinear and linear iterations

As a first aspect, the trade-off between nonlinear iterations and linear iterations as shown in Table 5.12 is considered. The GMRes algorithm solving the linear problems aims at a reduction of the linear residual by six orders of magnitude relative to the initial residual. If this condition is fulfilled the algorithm will stop, otherwise the linear solution process terminates after a maximal number of GMRes iterations which is an additional parameter. For most of the results in this section, especially at the end of the nonlinear convergence process, the second criterion is reached. It is expected that a reduced number of allowed GMRes iterations corresponds to an inferior quality of the obtained linear solution and will thus result in an increased number of outer nonlinear iterations. However, the reduced effort in the linear problems may yield an overall increase in efficiency, unless the linear solutions are inadequate in a way that it yields a dramatic decrease of the nonlinear efficiency. Eventually, if the linear solution process is stopped too early, the outer nonlinear iteration will diverge. Keeping all other parameters fixed, an optimal number of GMRes iterations can be determined depending on the problem.

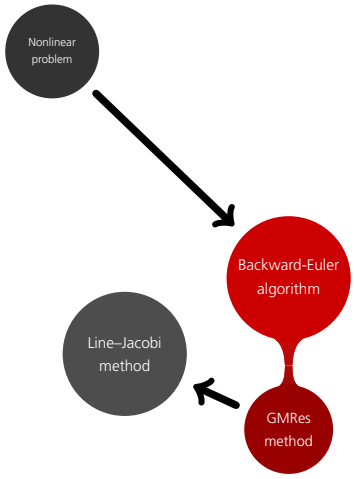

Figure 5.24: Critical solver elements for trade-off between nonlinear and linear iterations investigation are highlighted in red within the framework overview.

These theoretical considerations are confirmed by the numerical results for an h-startup strategy with structured meshes of the MDA test case from Section 5.1.2 with 33 728 elements and 8 432 elements, see Table 5.12 & 5.13, respectively. The solver for each of the meshes in this subsection is the Backward–Euler method with a GMRes algorithm preconditioned with a Line–Jacobi method. This solver approach is one of the most common single grid solvers in the DG community. The Line–Jacobi method is the smoother of the proposed linear multigrid algorithms and the Backward–Euler algorithm is the smoother for the nonlinear multigrid algorithms, therefore, all findings in this section will contribute to the efficiency of all proposed algorithms. Since these two algorithms are used in every computation of DG_ON.

It is possible to choose a different maximal number of GMRes iterations on each level of the startup strategy and, if used, the multigrid algorithms. Please note that for each computation shown in Table 5.12 the same maximal number of GMRes iterations per level is chosen. Since the nonlinear residual reduction on every level of the startup strategy is

no.	startup	levels	GMRes it.	CPU time	nonlinear iter.
1	10^{-6}	$h(4)$	100, 100, 100, 100	1.14	172, 104, 83, 119
2	10^{-6}	$h(4)$	80, 80, 80, 80	1.12	173, 106, 89, 161
3	10^{-6}	$h(4)$	60, 60, 60, 60	1.00	175, 109, 84, 166
4	10^{-6}	$h(4)$	40, 40, 40, 40	1.00	181, 116, 92, 219
5	10^{-6}	$h(4)$	20, 20, 20, 20	1.27	216, 144, 117, 425
6	10^{-6}	$h(4)$	10, 10, 10, 10	∞	325, 218, 148, ∞

Table 5.12: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 33 728 elements: influence of linear solver settings.

set to 10^{-6}, as opposed to 10^{-12} on the top level, the number of nonlinear iterations on these levels are not as widely spread as on the fine level. Additionally, on the top level of the startup strategy a much more of all nonlinear iterations per level reach the maximal number of GMRes iterations, due to the nonlinear residual tolerance of 10^{-12}.

It can be observed in Table 5.12 that if the maximal number of GMRes iterations is decreased the number of nonlinear iterations on the fine level increases. This is an anticipated effect as described above. However, the computation which can use the largest maximal number of GMRes iterations and has therefore the minimal number of nonlinear iterations is not the fastest computation. Hence, a trade-off between nonlinear and linear iterations must be made.

When monitoring the CPU time on the finest grid only the computation in Table 5.12 (row 3) takes 0.45 normalized CPU time, whereas the computation in Table 5.12 (row 1) with 100 maximal GMRes iterations takes 0.50 in normalized CPU time, even though the latter computation requires 119 nonlinear iterations only in comparison to 166 nonlinear iterations for the former computation on the fine grid and top level of the startup strategy. This indicates that for every level of the startup strategy an optimal number of GMRes iterations can be chosen.

In Table 5.13 parameter settings and results are shown for the MDA 30P30N test case on a structured mesh with 8 432 elements. Here, a two level p-startup strategy is performed and the same effect can be seen as in Table 5.12. Moreover, a maximal number of GMRes steps of 40 is chosen for the $p = 1$ level of the startup strategy for all simulations in Table 5.13. Hence, the effect of trading between the number of linear and nonlinear iterations can be examined on the top level, only.

Again, it can be observed that more nonlinear iterations do not necessarily result in an increased CPU time when in every nonlinear iteration less work is required. The difference in the number of nonlinear iterations on the top level is relatively small. This is due to the fact that at the end of the nonlinear solution processes the linear problems are more difficult to solve because of the increase in the pseudo time step size by the SER time step

no.	startup	levels	GMRes it.	CPU time	nonlinear iter.
1	10^{-6}	$p(2)$	40, 100	1.24	293, 449
2	10^{-6}	$p(2)$	40, 80	1.05	293, 451
3	10^{-6}	$p(2)$	40, 40	1.00	293, 478
4	10^{-6}	$p(2)$	40, 20	∞	293, ∞

Table 5.13: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 8 432 elements: influence of linear solver settings, see Figure 5.25.

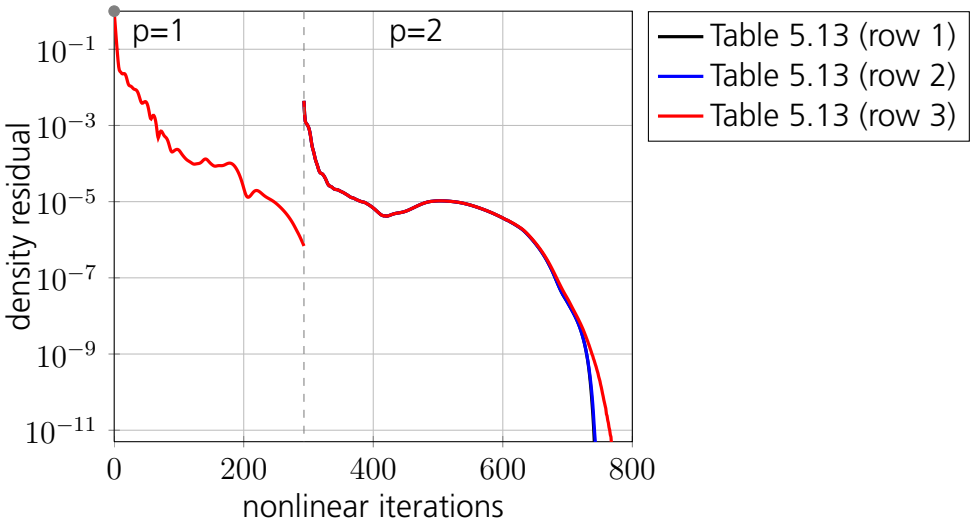

Figure 5.25: MDA 30P30N $p = 2$ $k\omega$-computation, from Table 5.13, on a structured mesh with 8 432 elements with a 10^{-6}-$p(2)$-startup strategy.

control of the Backward–Euler algorithm. Hence, the linear problems at the end of the nonlinear solution process are less diagonal dominant in comparison to the initial phase of the nonlinear solution process because of the vanishing mass matrix addition. Therefore, at the end of the solution process, near the asymptotic convergence, it is more important that the linear problems are solved well than at the start of the computation. Figure 5.25 shows that the convergence rates of the computations in Table 5.13 differ at the end of the nonlinear convergence, only.

This trade-off between linear and nonlinear iterations can be seen for all test cases computed in this work, see also [75], and will be used to improve the CPU run time.

Conclusion

An optimal number of GMRes iterations in terms of CPU time or nonlinear iterations can be determined depending on the problem, while keeping all other parameters fixed. In Sections 5.2.5 - 5.2.6 this optimal number of GMRes iterations is further investigated because it might change if linear multigrid preconditioners are applied. Furthermore, a computations with a minimal number of nonlinear iterations may not be the fastest computation. Hence, a trade-off between nonlinear and linear iterations must be made.

The Line–Jacobi method and the Backward–Euler algorithm are the two algorithms which are used in every computation of the proposed framework there the investigations of this section influence all other investigations in this chapter. For all test cases in this thesis a optimal number of GMRes iterations in terms of overall CPU time lies between 20 and 50 GMRes iterations depending on the test case, mesh, polynomial degree and level.

5.2.4 Coarse level linearization

The next aspect that requires some attention is the Galerkin transfer operator which is introduced in Section 4.6. Recall, that the Galerkin transfer is applied on the Jacobian matrix to generate lower level Jacobians instead of reassembling the Jacobian on the lower levels of a multigrid computation.

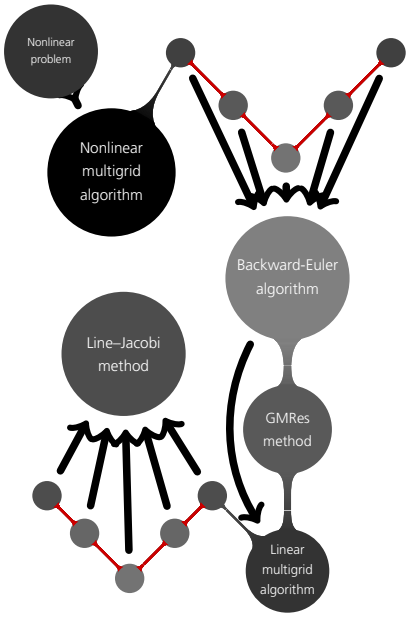

Figure 5.26: Critical solver elements for the coarse level linearization investigation are highlighted in red within the framework overview.

Different coarse level linearizations

The results in Tables 5.15 (rows 1 & 2) & 5.14 (rows 1 & 2) illustrate that the nonlinear convergence with the linear multigrid algorithm used as a preconditioner is almost unaffected by this choice. However, there is a notable difference in computational time. Tables 5.15 & 5.14 show that for the current implementation the projection of given fine level data, which in a sense is a scaled copy, to be more efficient in terms of CPU time than the recomputation of coarse level data.

The difference between reassembling the Jacobian on lower levels and using the Galerkin transfer operator is significantly larger in the case of a nonlinear multigrid algorithm. The only case where the Galerkin transfer gives a small improvement is the L1T2 test case on the unstructured WUT mesh, see Table 5.14 (rows 5 & 6). Note that, for this case a nonlinear p-multigrid algorithm is applied. For a nonlinear h-multigrid algorithm even the robustness of the proposed algorithm seems to suffer if the Galerkin transfer is not used.

The p-multigrid algorithm cases in Table 5.16 (rows 4 & 5) show this lack of robustness even for the linear multigrid algorithm. Furthermore, the combination of a nonlinear and linear p-multigrid algorithm, as shown in Table 5.16 (s 8 & 9), can not recover from these negative effects.

Conclusion

In addition to the gain in computational time it also is much more stable and robust to use the Galerkin transfer operator. Since the DG discretization yields a Galerkin type discretization also on the coarse levels, the difference between the two approaches might seem surprisingly large. In fact, the only differences arise from the treatment of some boundary conditions, the viscous flux terms associated with inter-element discontinuities

no.	startup	NMG	LMG	levels	coarse lin.	CPU time	nonlinear iter.
1	1	-	/	$h(3)$	GT	1.00	1, 1, 295
2	1	-	/	$h(3)$	redisc.	1.19	1, 1, 284
3	1	v	-	$h(3)$	GT	0.67	1, 1, 228
4	1	v	-	$h(3)$	redisc.	NaN	1, 1, NaN
5	1	v	-	$p(2)$	GT.	0.81	1, 1, 340
6	1	v	-	$p(2)$	redisc.	0.88	1, 1, 338

Table 5.14: L1T2 $(p = 2)$ SA-computation on the unstructured WUT mesh: effect of coarse level linearization.

no.	startup	NMG	LMG	levels	coarse lin.	CPU time	nonlinear iter.
1	10^{-6}	-	/	$p(2)$	GT	1.00	91, 58
2	10^{-6}	-	/	$p(2)$	redisc.	1.19	91, 58
3	10^{-6}	v	-	$p(2)$	GT	0.84	91, 66
4	10^{-6}	v	-	$p(2)$	redisc.	1.04	91, 78

Table 5.15: VFE-2 $(p = 2)$ $k\omega$-computation on a structured mesh with 13 816 elements: effect of coarse level linearization.

and the fact that the Galerkin operator always employs a linearization about the current fine level solution. The believe is that the latter effect is dominant, i. e. it is beneficial to avoid using a restricted coarse approximation of the fine level solution as linearization point for the Jacobian.

In all of the computations shown in this section the Galerkin transfer matrices were computed everytime they where used. An additional reduction in terms of CPU time can be achieved if the Galerkin transfer matrices are stored for each level. This is a typical trade-off between memory consumption and CPU time. When this framework is compared to other CFD codes the Galerkin transfer is stored to decrease the CPU time, see Section 5.3.

All multigrid computations in other sections employ the Galerkin transfer operator as a "best practice" approach. Similar findings where also published in [75].

no.	startup	NMG	LMG	levels	coarse lin.	CPU time	nonlinear iter.
1	10^{-6}	-	-	$h(4)$	-	1.00	216, 144, 117, 425
2	10^{-6}	v	-	$h(4)$	GT	0.89	216, 106, 74, 242
3	10^{-6}	v	-	$h(4)$	redisc.	1.61	216, 110, 92, 334
4	10^{-6}	-	v	$p(2)$	GT	1.39	640, 622
5	10^{-6}	-	v	$p(2)$	redisc.	NaN	640, NaN
6	10^{-6}	v	-	$p(2)$	GT	1.65	640, 906
7	10^{-6}	v	-	$p(2)$	redisc.	NaN	640, NaN
8	10^{-6}	v	/	$p(2)$	GT	1.48	640, 740
9	10^{-6}	v	/	$p(2)$	redisc.	NaN	640, NaN

Table 5.16: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 33 728 elements: effect of coarse level linearization.

5.2.5 Linear multigrid algorithm as a preconditioner

This section discusses various preconditioners applied in the GMRes algorithm. Here, the main focus lies on the effect on the number of nonlinear iterations and CPU time.

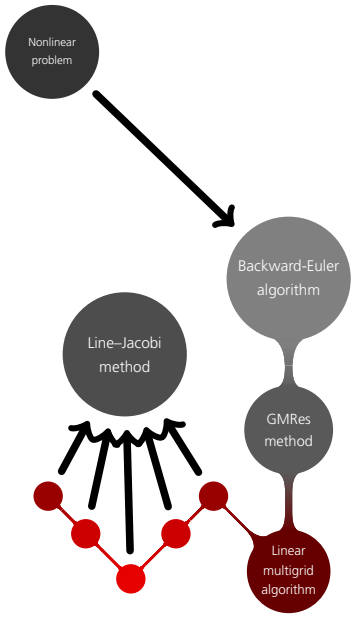

Figure 5.27: Critical solver elements for the investigation of the GMRes preconditioner are highlighted in red within the framework overview.

Cycle type investigation

As has been seen in Section 5.2.3 it is important how accurate the linear problems are solved in order to achieve a stable and fast nonlinear solver. With this in mind several different linear multigrid cycle types are suggested as a preconditioner to improve the overall solution process.

In Table 5.17 an h-startup strategy is used to compute a $p = 2$ solution for the MDA 30P30N test case on a structured mesh with 33 728 elements. The first, second, third and sixth row of Table 5.17 correspond to a line preconditioned GMRes method without a multigrid algorithm applied. The overall computations do not converge if less than 20 GMRes iterations are allowed on each level of the startup strategy. In that case the linear problems are not sufficiently solved. Note that the computations in Table 5.17 (rows 1,2,6) are the same as in Table 5.12 (rows 3,5,6).

no.	startup	LMG	levels	GMRes it.	CPU time	nonlinear iter.
1	10^{-6}	-	$h(4)$	60 60 60 60	1.00	175, 109, 84, 166
2	10^{-6}	-	$h(4)$	20 20 20 20	1.27	216, 144, 117, 425
3	10^{-6}	-	$h(4)$	20 20 20 120	1.07	216, 144, 117 108
4	10^{-6}	/	$h(4)$	20 20 20 20	0.93	216, 113, 76, 132
5	10^{-6}	v	$h(4)$	20 20 20 20	0.95	216, 108, 82, 110
6	10^{-6}	-	$h(4)$	10 10 10 10	∞	325, 218, 148, ∞
7	10^{-6}	/	$h(4)$	10 10 10 10	1.29	325, 139, 113, 269
8	10^{-6}	v	$h(4)$	10 10 10 10	1.02	325, 123, 78, 145
9	10^{-6}	w	$h(4)$	10 10 10 10	1.04	325, 123, 82, 119
10	10^{-6}	v	$h(4)$	20 10 10 10	0.93	216, 123, 78, 150

Table 5.17: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 33 728 elements: effect of the linear multigrid preconditioner.

In the lower part of Table 5.17 one can see the effect from enhancing the preconditioner with the help of lower level discretizations. Using one cycle of a linear h-multigrid algorithm as preconditioner for the GMRes algorithm stabilizes the overall solution process. Note, that because the Line–Jacobi method is used as a smoother in the linear h-multigrid algorithm, the only difference between the computations in Table 5.17 is the additional use of lower level discretizations in the preconditioner step of the GMRes iterations. Furthermore, in each test case in this section the number of Line–Jacobi steps and the number of smoother steps in the linear multigrid algorithm are chosen identical. Finally, all other solver settings except of the choice of a linear multigrid cycle are the same in this section, unless mentioned otherwise.

In order to achieve convergence of the nonlinear problem with the Backward–Euler algorithm and only the Line–Jacobi method as a preconditioner on the mesh with 33 728 elements the maximal number of GMRes iterations has to be increased at least to a value of 20, see Table 5.17 (rows 1,2,3,6). This results in a corresponding increase in the CPU time of 0.25 in comparison to using a linear h-multigrid preconditioner V-cycle with maximal 10 GMRes iterations, see Table 5.17 (rows 2 & 8). Moreover, when using a linear h-multigrid preconditioner and at most 20 GMRes iterations on every level, see Table 5.17 (rows 4 & 5), the CPU time is reduced even below the best version of the Backward–Euler algorithm without a linear multigrid algorithm, Table 5.17 (row 1).

The results above show that the linear h-multigrid algorithm has a positive effect on the nonlinear convergence. If a maximal number of GMRes iterations is provided it is definitely reached at the end of the convergence history for all test cases in this work. The linear problems are much harder to solve at the end of the nonlinear convergence history because of the large pseudo-time step size provided by the SER time step control. There, the linear multigrid algorithms provide the most assistance to solve the linear problems

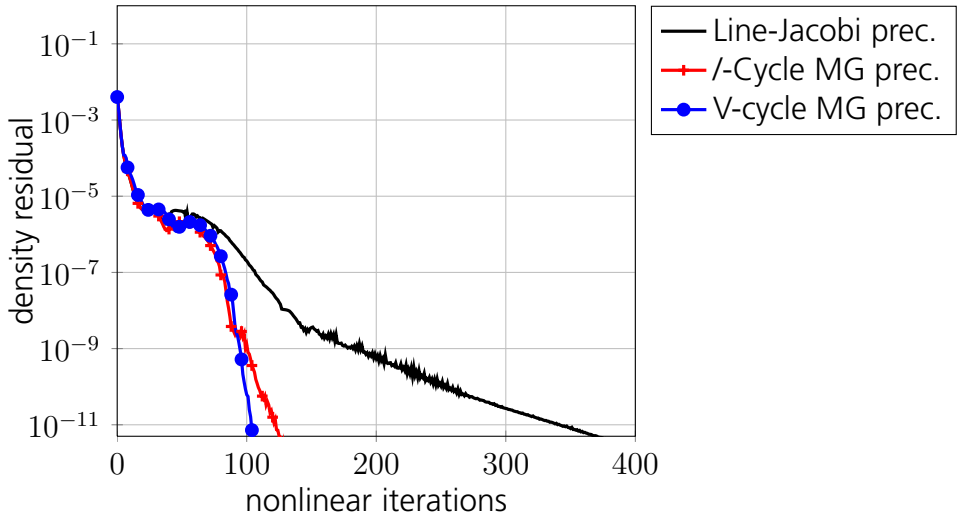

Figure 5.28: MDA 30P30N $p = 2$ $k\omega$-computation on a structured mesh with 33 728 elements, from Table 5.17 (rows 2,4,5): The top level of a 10^{-6}-$h(4)$-startup strategy.

more accurately. This is due to the fact, that at the beginning of the convergence history the linear problems are easy to solve, and therefore, the maximal number of GMRes iterations is not reached before the linear residual is reduced 6 orders of magnitude which is required for these test cases. At the end of the nonlinear convergence regime for a given number of maximal GMRes iterations a better nonlinear convergence rate can be observed for all test cases when a stronger preconditioner is used, to illustrate this fact see Figure 5.28. In this figure the linear problems are only solved with maximal 20 GMRes iterations or a relative reduction of 6 orders of magnitude.

As soon as the maximal number of linear iterations is reached in each nonlinear iteration of the computations shown the convergence rates differ. The unmarked line represents a Line–Jacobi preconditioned computation, whereas the line marked with a "+" represents a linear multigrid preconditioned result applying a saw tooth cycle and the line marked with "•" a V-cycled linear multigrid preconditioned computation. Here the computation with the strongest cycle type yields the best results. Note that the linear multigrid algorithm helps to establish a satisfying convergence rate at the end of the nonlinear convergence regime where asymptotic convergence should be expected. In order to achieve a similar convergence rate and number of nonlinear iterations on the top level of the startup strategy as the computation in Table 5.17 (row 5), which is also the marked line with a "•", for a computation where no multigrid preconditioner is used the maximal number of GMRes iterations has to be set to 120, see Table 5.17 (row 3). The time spent on the top level for such a computation is 0.54 in comparison to the time the computation in Table 5.17 (row 5) spends on the top level of the startup strategy which is 0.37 normalized CPU time. This shows that the linear multigrid preconditioner is beneficial in terms of CPU time and cost per nonlinear iteration if the same convergence rates are intended. The linear multigrid preconditioner enables us to reduce the maximal number of GMRes iterations drastically

without a deteriorate effect on the number of nonlinear iterations or convergence rates. Moreover, since a wider range of maximal numbers of GMRes iterations can be chosen without the lose of nonlinear convergence the linear multigrid preconditioner improves the robustness of the solution process.

In Table 5.17 (rows 5-8) 10 GMRes iterations were made on every level. The findings in Section 5.17 indicate that different GMRes iterations could be optimal for each level in terms of CPU time, e. g. on the lowest level of the startup strategy and therefore on the coarsest grid because there the linear multigrid algorithm is not active. Hence, a fair comparison in computational time between Table 5.17 (rows 2-4) and Table 5.17 (rows 5-8) can only be made if the startup strategy levels where no multigrid algorithm can be used are solved with the same settings. This is accomplished in the last row of Table 5.17, here 20 GMRes iterations are made on the coarsest level. As can be seen by comparing row 8 and the last row the gain while solving this level with 20 GMRes iterations is about 0.09 in normalized CPU time. The computation is even faster than when applying a maximal number of GMRes iterations of 20 on every level with the same solver settings, see Table 5.17 (row 5).

no.	startup	LMG	levels	GMRes it.	CPU time	nonlinear iter.
1	10^{-6}	-	$p(2)$	40 40	1.00	293, 478
2	10^{-6}	-	$p(2)$	40 20	∞	293, ∞
3	10^{-6}	-	$p(2)$	40 10	∞	293, ∞
4	10^{-6}	v	$p(2)$	40 10	0.90	293, 492

Table 5.18: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 8 432 elements: effect of linear multigrid preconditioner.

In Table 5.18 a similar effect is observed for the linear p-multigrid algorithm. By reducing the maximal number of linear iterations on the top level to 10, where the linear multigrid algorithm is active, and applying a V-cycle as a preconditioner a reduction in CPU time can be observed. When only 10 GMRes iterations are allowed and using the Line–Jacobi preconditioner only the computation did not converge, Table 5.18 (row 3).

no.	startup	LMG	levels	ave. GMRes it.	CPU time	nonlinear iter.
1	1	-	-	456	1.00	1, 1, 153
2	1	/	$h(3)$	196	0.26	1, 1, 153
3	1	v	$h(3)$	115	0.15	1, 1, 153
4	1	w	$h(3)$	103	0.14	1, 1, 153

Table 5.19: MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 33 728 elements: effect of linear multigrid preconditioner.

An investigation is presented in Table 5.19 where the linear problems are solved with a relative residual reduction of 10 orders of magnitude. Obviously, since all linear problems

are solved with the same accuracy the nonlinear iterations do not differ. Recall that in all other computations where GMRes iterations are reported the residual is reduced by 6 orders of magnitude or until the number of the prescribed GMRes iterations is reached, see Section 5.2.1. In contrast to that, here in Table 5.19 the linear solver will stop only if the linear residual is reduced 10 orders of magnitude relative to the starting residual.

It is necessary to mention that in Table 5.19 only a one step startup strategy is used. Therefore the CPU time reflects nearly the whole nonlinear convergence history on the top level. Moreover, nearly no conclusion can be drawn from Table 5.19 in terms of robustness of the applied solver algorithms. Instead, the advantage in CPU time can be observed, since the linear problems are always reduced 10 orders of magnitude relative to the starting residual. Using only the Line–Jacobi method without a linear multigrid algorithm yields by far the slowest CPU time, Table 5.19 (row 1). The chosen multigrid cycles are applied once per GMRes iteration and four smoother steps are employed. This means that for a sawtooth-cycle on the top, mid and low level four post-smoothing steps are applied but no pre-smoothing steps, whereas for the V- and W-cycle four pre- and four post-smoothing steps are preformed. Hence, for more complex cycle types more smoother steps are applied on each level if the number of smoother steps is constant on every level as described above. Therefore, the effect of how to conduct a level sequence in a cycle and more importantly how many smoother steps on each level are beneficial can be seen in Table 5.19. As can be seen, the computation with the smallest average number of GMRes iterations results in the fastest CPU time. Applying more complex cycle types and smoother steps as preconditioner and, therefore, adding more CPU time and cost to the preconditioner is beneficial in this case because the overall CPU time reduces. Obviously this is due to the fact of the smaller average number of GMRes iterations.

Coarse level solving

Until now in all computations in this sections the same number of smoother steps are performed within the linear multigrid **Algorithm 3**. Even on the lowest level the same number of Line–Jacobi iterations are applied as a solver. Solving the linear problem on the lowest level more accurately should be beneficial, and therefore, reduce the number of average GMRes iterations further. The same computation as in Table 5.19 (row 3) has been preformed but with ten times more smoother steps on the lowest level of the linear multigrid preconditioner than on the other levels, see Table 5.21 (row 2). Hence, the computation in Table 5.19 (row 3) is the same as in Table 5.21 (row 1). The average number of GMRes iterations in the computation in Table 5.21 (row 2) is 103 in comparison to 115 for the computation in Table 5.19 (row 3). Even though the reduction of the average number of GMRes iterations is only 12 steps a faster CPU time can be observed. Reducing the average GMRes iterations by a better solved coarse level problem in the linear multigrid preconditioner is beneficial, and therefore, aligns with the common multigrid theory. The

CPU time of the computation with 10 times more smoother steps on the lowest level and a V-cycle, Table 5.21 (row 2), is as fast as the computation with a W-cycle and the same number of smoother steps on every level, Table 5.19 (row 4). A 8% reduction in CPU time can be observed for this test case when only the effect of solving the lowest level more accurately is taken into account and using the same linear multigrid cycle. For this reason further investigations on solving the lowest level problem more accurately are conducted.

no.	startup	LMG	levels	ave. GMRes it.	CPU time	nonlinear iter.
1	1	v (4 4 4)	$h(3)$	89	1.00	1, 1, 63
2	1	v (40 4 4)	$h(3)$	67	0.93	1, 1, 63

Table 5.20: MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 8 432 elements: comparison of the low level solver within the multigrid preconditioner.

no.	startup	LMG	levels	ave. GMRes it.	CPU time	nonlinear iter.
1	1	v (4 4 4)	$h(3)$	115	1.00	1, 1, 153
2	1	v (40 4 4)	$h(3)$	103	0.92	1, 1, 153

Table 5.21: MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 33 728 elements: comparison of the low level solver within the multigrid preconditioner.

In Tables 5.20 & 5.21 the number of average GMRes steps can be reduced by increasing the number of smoother steps on the lowest level. In both tables the linear residual is reduced relative to the initial residual by 10 orders of magnitude without providing a maximal number of GMRes iterations. However, solving the lowest level of the linear multigrid more accurately is not beneficial for every test case in terms of CPU time, see Tables 5.22 to 5.24. This is due to the fact, that the time which is reduced by performing a smaller number of average GMRes iterations is spent by solving the lowest level of the linear multigrid preconditioner with 5 times more steps. It follows that the factor 5 might not be appropriate for every test case and mesh.

For the test cases where the linear problems are solved using only a relative residual reduction of 6 orders of magnitude or a maximal number of GMRes steps, whatever is reached first, the same effects can be observed, see Tables 5.22 to 5.24. The average number of linear solving steps can be reduced but a reduction in CPU time is not always achieved. One could argue that for an appropriate number of coarse level smoother steps even in the cases in Tables 5.22 to 5.24 where currently no reduction in CPU time is seen a satisfying result could be accomplished, e. g. by multiplying the smoother steps on lowest level of the linear multigrid preconditioner by a factor different from 5. Moreover, the maximal number of GMRes iterations can be varied to reduce the overall CPU time as seen in Section 5.2.3. Another fact which aligns with the multigrid theory is that weaker cycle types e. g. where less work is done on the lower levels, result in a higher number of average GMRes iterations, see Table 5.24.

no.	startup	LMG	levels	GMRes it.	ave. it.	CPU time	nonlinear iter.
1	10^{-6}	v (4 4 4)	$h(3)$	40 40 40	32	1.00	199, 101, 56
2	10^{-6}	v (40 4 4)	$h(3)$	40 40 40	12	1.18	193, 104, 56

Table 5.22: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 8 432 elements: comparison of the low level solver within the multigrid preconditioner.

no.	startup	LMG	levels	GMRes it.	ave. it.	CPU time	nonlinear iter.
1	10^{-6}	v (4 4 4 4)	$h(4)$	20 10 10 10	15	0.85	216, 123, 78, 150
2	10^{-6}	v (40 4 4 4)	$h(4)$	20 10 10 10	9	0.82	171, 104, 78, 124

Table 5.23: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 33 728 elements: comparison of the low level solver within the multigrid preconditioner.

no.	startup	LMG	levels	GMRes it.	ave. it.	CPU time	nonlinear iter.
1	10^{-6}	/ (4 4)	$p(2)$	20 20	15	1.00	91, 56
2	10^{-6}	/ (40 4)	$p(2)$	20 20	8	1.05	91, 54
3	10^{-6}	v (4 4)	$p(2)$	20 20	14	1.02	91, 55
4	10^{-6}	v (40 4)	$p(2)$	20 20	7	1.07	91, 54
5	10^{-6}	/ (4 4)	$p(2)$	10 10	9	1.06	91, 64
6	10^{-6}	/ (40 4)	$p(2)$	10 10	6	1.03	91, 55
7	10^{-6}	v (4 4)	$p(2)$	10 5	7	1.14	91, 71
8	10^{-6}	v (40 4)	$p(2)$	10 5	4	1.01	91, 55

Table 5.24: VFE-2 $(p = 2)$ $k\omega$-computation on a structured mesh with 13 816 elements: comparison of the low level solver within the multigrid preconditioner.

In Table 5.25 a three level linear p-multigrid algorithm is applied for a $p = 3$ computation. The findings from the previous Tables 5.22-5.24 where incorporated by choosing a higher number of smoother steps on the lowest level. For the single level solver the computation did not converge but if a linear multigrid is applied the computations converge. Moreover, a W-cycle in the p-multigrid algorithm case is also more beneficial in terms of nonlinear iteration. This was also observed for the h-multigrid algorithm with the same number of levels. It shows that, the number of nonlinear iterations can be reduced by enhancing the preconditioner with stronger multigrid cycles. These multigrid cycles are stronger because of the different number of smoother steps per level. In fact with the solver settings given in Table 5.25 (row 2,4,5) the saw tooth cycle performs post-smoothing steps only with four Line-Jacobi iterations applied on the mid level, whereas for a V-cycle it is eight and for a W-cycle it is twelve on the mid level. The computation in Table 5.25 (row 3) based on a saw tooth cycle performs the same number of smoother steps on every level as the computation in Table 5.25 (row 4) which uses a V-cycle. The only difference is at which point the lower level approximations are solved within the linear multigrid algorithm. This computation requires the same number of nonlinear iterations

no.	startup	LMG	levels	GMRes it.	CPU time	nonlinear iter.
1	1	- (4)	$p(3)$	20 20 20	∞	1, 1, ∞
2	1	/ (12 4 4)	$p(3)$	20 20 20	1.00	1, 1, 197
3	1	/ (12 8 8)	$p(3)$	20 20 20	1.06	1, 1, 175
4	1	v (12 4 4)	$p(3)$	20 20 20	1.10	1, 1, 175
5	1	w (12 4 4)	$p(3)$	20 20 20	1.15	1, 1, 162

Table 5.25: L1T2 $(p = 3)$ SA-computation on the unstructured CEN mesh: effect of the linear multigrid preconditioner.

to converge as the computation where a V-cycle is used, but a slight reduction in CPU time can be observed by 0.04 normalized CPU time. Hence, the point at which the lower level problems are addressed is less important than the number of overall smoother steps per level, since the nonlinear convergence history stays the same for both computations, see Table 5.25 (rows 3 & 4).

Finally, note that no difference in algorithmic behavior can be observed for $p = 3$ computations. In terms of CPU time the more costly multigrid cycles, Table 5.25 (rows 4-5), perform slightly worse than the cheapest one, Table 5.25 (row 2). Hence, the reduction in the number of nonlinear iterations is not large enough to counterbalance the more costly cycle type, similar effects could be seen before for other test cases or meshes.

Linear multigrid preconditioner conclusion

For all results shown the benefit of allowing a reduced number of GMRes iterations while using a linear h-multigrid as a preconditioner carries over to a reduction in the overall CPU time. While using the same solver setting on every level the positive effects of this method are a gain in both stability and nonlinear convergence. Of course, variation of the solver settings per level is a legitimate way to further enhance the solver robustness and efficiency. Another important point of this investigation is that there is no difference in content between the p- or h-linear multigrid preconditioner, all statements from above hold for both linear multigrid types. Furthermore, these findings also hold true for both turbulence models considered.

All results shown in this section are obtained by only performing one multigrid cycle as a preconditioner. However, in this framework, it is possible to choose as many linear multigrid cycles as desired. In the "best practice" approach only one preconditioner cycle is applied in each GMRes iteration, since this approach showed satisfying results in terms of CPU time or nonlinear convergence rate for all test cases and meshes, even in comparison with other CFD codes see Section 5.3.

Following from the findings in this section and Section 5.2.3 there should exist an optimal combination of maximal number of GMRes iterations and the number of linear multigrid

preconditioner cycles for each test case in terms of CPU time or nonlinear convergence rate.

Moreover, the "best practice" approach will always perform 1.5 to 2 times the number of smoother steps on the lowest level than on the other levels of the multigrid preconditioner. This is due to the fact that a factor of 5 is often not beneficial in terms of overall CPU time but always in terms of nonlinear iterations. The smaller factors of the "best practice" approach show a reduction of CPU time and nonlinear iterations for all test cases and meshes.

Please note that the outcome of the investigation of the linear multigrid algorithms as a preconditioner is similar to findings published in [75].

5.2.6 Linear multigrid algorithm as a solver

In this section the potential of the linear multigrid algorithm as a solver for the linear problems is discussed. These investigations approach the linear solution process from the other end of the algorithmic design in comparison to Section 5.2.5. Hence, a comparison between the approach to enhance the GMRes iterations with a linear multigrid algorithm versus a linear solver without using the GMRes algorithm at all is shown.

Recall from Section 4.2.2 that the GMRes method is a Krylov subspace method, and therefore an optimality condition is derived for the norm of the residual in the Krylov subspace. The linear multigrid algorithm applied as a solver lacks such a condition. When the GMRes algorithm is combined with one linear multigrid preconditioner cycle, it is superior to the GMRes algorithm with only the Line–Jacobi method as a Preconditioner, as shown in Section 5.2.5. Therefore, this algorithm will be the base line for comparison. This solver which solves a minimization problem in a subspace in combination with an algorithm which uses an additional coarse level as correction should be superior to a solver which only uses additional coarse levels in an iterative scheme as correction. Following from this, the GMRes algorithm with a linear multigrid preconditioner should provide better results in terms of robustness and average linear iterations than the sole linear multigrid solver. However, the linear multigrid solver should provide better results than the Line–Jacobi method employed as a linear solver.

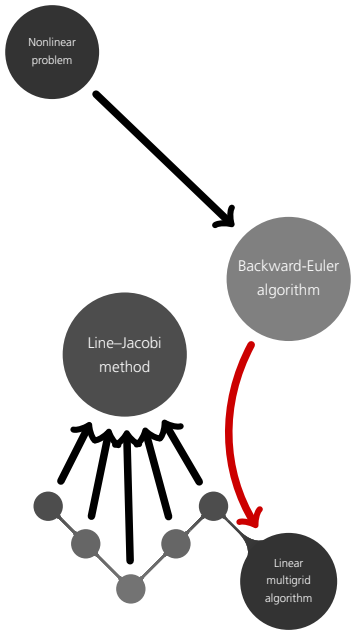

Figure 5.29: Critical solver elements for the linear solver investigation are highlighted in red within the framework overview.

Linear solver choice

In Table 5.26 the linear problems are solved to a relative linear residual of ten orders of magnitude and no limit for the number of linear iterations is given.

The computation in Table 5.26 (row 1) needs at most 178 GMRes iterations, preconditioned with a linear multigrid algorithm, to reduce the relative linear residual by ten orders of magnitude. In comparison, the linear multigrid applied as a linear solver, Table 5.26 (row 2), needs up to 657 linear iterations for the same order of reduction. Moreover, the average number of GMRes iterations of the computation in Table 5.26 (row 1) is 89 in contrast to an average of 253 linear multigrid V-cycles for the computation in Table 5.26

no.	startup	LMG	levels	ave. LMG it.	ave. GMRes it.	CPU time	nonlinear iter.
1	-	v (4 4 4)	$h(3)$	89	89	1.00	1, 1, 63
2	-	v (4 4 4)	$h(3)$	253	0	2.13	1, 1, 63
3	-	v (20 4 4)	$h(3)$	192	0	1.94	1, 1, 63
4	-	v (40 4 4)	$h(3)$	184	0	2.31	1, 1, 63

Table 5.26: MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 8 432 elements: influence of different linear solver algorithms.

(row 2). Thus, it follows that applying the linear multigrid as the linear solver is not satisfying in terms of CPU time and iterations required to reduce the relative linear residual by 10 orders of magnitude. Some enhancement in CPU time can be observed if the coarse level problems in the V-cycle of the linear multigrid solver are solved more accurately, see Table 5.26 (row 3). However, if the number of coarse level smoother steps is set too high it can have a negative effect on the overall CPU time, see Table 5.26 (row 4). Please note that in row one of Table 5.26 the average number of linear multigrid cycles is equal to the average number of GMRes iterations. This is due to the fact that in every GMRes iteration one linear multigrid preconditioner cycle is performed as described in the beginning of this section.

Even though the linear problems in every nonlinear iteration are usually not solved that accurately, this investigation shows the weakness of this particular solver because of the higher average amount of work in every nonlinear iteration. This observation is also supported by Table 5.27 even though in this case a maximal number of linear iterations is prescribed and the maximal order of relative residual reduction is 10^{-6} as usual.

no.	startup	LMG	levels	LMG it.	GMRes it.	CPU time	nonlinear iter.
1	10^{-6}	v	$h(3)$	40 40 40	40 40 40	1.00	199, 101, 56
2	10^{-6}	v	$h(3)$	40 40 40	0 0 0	1.07	214, 104, 62
3	10^{-6}	-	$h(3)$	40 40 40	0 0 0	NaN	214, 126, NaN
4	10^{-6}	v	$h(3)$	20 20 20	20 20 20	0.90	213, 107, 69
5	10^{-6}	v	$h(3)$	20 20 20	0 0 0	NaN	249, 116, NaN

Table 5.27: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 8 432 elements: influence of different linear solver algorithms.

In terms of robustness the linear multigrid algorithm is also worse than the GMRes algorithm in combination with a linear multigrid preconditioner, see Table 5.27 (rows 4 & 5). Table 5.27 (rows 2 & 3) show that the linear multigrid algorithm is more robust than a Line–Jacobi method as a linear solver, as expected. Here, the additional work on lower level approximations is beneficial in terms of robustness as anticipated above.

no.	startup	LMG	levels	LMG it.	GMRes it.	CPU time	nonlinear iter.
1	10^{-6}	/ (4 4)	$p(2)$	20,20	20,20	1.00	91, 56
2	10^{-6}	/ (4 4)	$p(2)$	20,20	0,0	2.05	91, 142
3	10^{-6}	v (40 4)	$p(2)$	20,20	20,20	1.07	91, 54
4	10^{-6}	v (40 4)	$p(2)$	20,20	0,0	1.07	91, 54

Table 5.28: VFE-2 $(p = 2)$ $k\omega$-computation on a structured mesh with 13 816 elements: effect of coarse level linearization.

For the linear p-multigrid similar findings can be observed. If a weaker linear multigrid cycle is chosen, disappointing results are achieved, Table 5.28 (row 2), but if a stronger cycle is applied the results in terms of overall CPU time are similar, see Table 5.28 (row 4).

Conclusion

The findings in this section indicate that the linear multigrid can be applied as a linear solver. Setting up this particular solver seems to require much more tuning by the user to create satisfying results in terms of overall CPU time. Moreover, it seems less robust than the GMRes method with a linear multigrid preconditioner, see Table 5.26.

Finally, it seems more effective to apply a minimization problem in a subspace preconditioned by an algorithm which uses additional coarse levels instead of only using a Krylov method or an iterative multigrid solver. Therefore the "best practice" to solve the linear problems will be a GMRes method with a linear multigrid preconditioner, as described in Section 5.2.5.

5.2.7 Investigation of the nonlinear multigrid algorithm

This section is similar to Section 5.2.5 & 5.2.6 in terms of investigations conducted. However, here the focus lies on the nonlinear multigrid algorithm. Note that, the combination of the linear and nonlinear multigrid algorithm will not be used in this section, see Figure 5.30. The sole purpose of this section is to emphasize the beneficial contribution of the nonlinear multigrid algorithm to the framework.

The Backward–Euler algorithm combined with an SER time step control achieves asymptotic convergence rates at the end of the solution process. This is due to the fact that the Backward–Euler algorithm gets close to a Newtons method for large pseudo time steps. Such a characteristic is desirable for all solver algorithm. Hence, it is desired to maintain this characteristic also in the nonlinear multigrid algorithm. As discussed in Section 4.2, the Backward–Euler algorithm will be the smoother choice in the nonlinear multigrid solver.

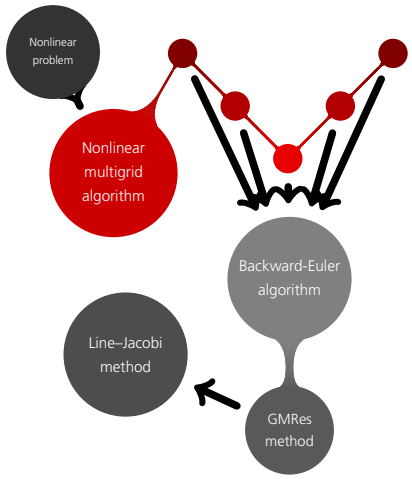

Figure 5.30: Critical solver elements for the investigation of the nonlinear multigrid algorithm are highlighted in red within the framework overview.

This smoother choice makes comparisons between the nonlinear multigrid algorithm and the single-grid Backward–Euler solver particularly simple. The single grid Backward–Euler solver can be seen as a one level multigrid algorithm. In this case the top level smoothing steps correspond to nonlinear Backward–Euler iterations. Therefore, a comparison of top level smoothing steps against nonlinear single-grid iterations is just a comparison as to how many Backward–Euler iterations are performed on the top level. Recall that between the Backward–Euler iterations in the nonlinear multigrid algorithm additional work is performed on the lower levels, which is neglected by a comparison of top level nonlinear Backward–Euler iterations. However, the overall comparison is given in terms of CPU time shown in the last colum of all tabels in this section. In fact, a reduction of top level iterations is reflected in CPU time only if the reduction is sufficiently large to cope with the additional iterations conducted on lower levels. Which cycle type to choose will be discussed next.

Investigation of the cycle type

In Tables 5.29 & 5.30 one smoother step is performed every time the nonlinear multigrid **Algorithm 1** performs smoother steps in the chosen nonlinear multigrid cycle. Moreover, on the lowest level of the multigrid cycles also one smoother step is applied.

no.	startup	NMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	$h(3)$	1.00	19, 76, 300
2	10^{-6}	v	$h(3)$	0.87	19, 62, 194
3	10^{-6}	w	$h(3)$	0.76	19, 62, 154
4	10^{-6}	-	$p(2)$	1.07	336 310
5	10^{-6}	v	$p(2)$	1.39	336 300

Table 5.29: L1T2 $(p = 2)$ $k\omega$-computation on the unstructured CEN mesh: nonlinear multigrid cycle type investigation.

no.	startup	NMG	levels	CPU time	nonlinear iter.
1	1	-	$h(3)$	1.00	1, 1, 442
2	1	v	$h(3)$	0.69	1, 1, 232
3	1	w	$h(3)$	0.59	1, 1, 190
4	1	-	$p(2)$	0.75	1, 318
5	1	v	$p(2)$	0.73	1, 270

Table 5.30: L1T2 $(p = 2)$ SA-computation on the unstructured CEN mesh: nonlinear multigrid cycle type investigation.

For the h-multigrid computations in Tables 5.29 & 5.30 for both turbulence models exactly the same solver settings are chosen, except for the startup strategy. In both tables the number of nonlinear Backward–Euler iterations on the top level reduces significantly if the nonlinear multigrid algorithm is applied. Moreover, with the choice of a more costly cycle type such as the W-cycle even more nonlinear top level iterations can be saved. Since only one smoother step is applied in every step of the multigrid cycle, the entries in the colum "nonlinear iter." are two times the number of cycles in the nonlinear multigrid computations on the top level of a startup strategy. Therefore, in this column additional work which is conducted between the smoother steps on the top level can not be seen in case of a nonlinear multigrid computation, as mentioned at the beginning of this section. However, this additional work is reflected in the overall CPU time. For the computations in Tables 5.29 & 5.30 also a clear reduction in overall CPU time can be seen. Obviously, the factor between two computations regarding the number of nonlinear top level iterations from the last column stays not the same when comparing the overall CPU times.

Tables 5.31 - 5.34 show that the reduction in nonlinear fine level iterations is in some cases not large enough to compensate the additional work on the lower levels in terms of overall CPU time for the nonlinear multigrid computations. Recall that the cost of the

no.	startup	NMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	$h(4)$	1.00	175, 109, 84, 166
2	10^{-6}	v	$h(4)$	1.12	175, 82, 62, 130

Table 5.31: MDA 30P30N $(p = 2)$ $k\omega$-computation on a structured mesh with 33 728 elements: nonlinear multigrid cycle type investigation.

no.	startup	NMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	$h(3)$	1.00	72, 955, 75
2	10^{-6}	v	$h(3)$	0.63	72, 526, 50
3	10^{-6}	w	$h(3)$	0.63	72, 526, 44

Table 5.32: MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 134 912 elements: nonlinear multigrid cycle type investigation.

no.	startup	NMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	$p(2)$	1.00	84, 73
2	10^{-6}	v	$p(2)$	1.01	84, 70

Table 5.33: VFE-2 $(p = 2)$ $k\omega$-computation on a structured mesh with 110 528 elements: nonlinear multigrid cycle type investigation.

no.	startup	NMG	levels	CPU time	nonlinear iter.
1	1	-	$h(3)$	1.00	1, 1, 275
2	1	v	$h(3)$	0.78	1, 1, 142
3	1	w	$h(3)$	NaN	1, 1, NaN
4	1	v	$p(3)$	0.75	1, 1, 152
5	1	w	$p(3)$	0.78	1, 1, 140

Table 5.34: L1T2 $(p = 3)$ SA-computation on the unstructured CEN mesh: nonlinear multigrid cycle type investigation.

whole computation, including all startup strategies and lower level iterations can only be addressed by the CPU time provided in these tabels.

Moreover, the nonlinear iteration reduction is not particularly large for the p-multigrid algorithm in Table 5.33 in comparison to the single grid solver. The same effect is shown in Table 5.30 in comparison to the h-multigrid algorithms with the same solver settings. This could relate with the depth of the nonlinear multigrid algorithms since for the p-multigrid algorithm computations in Tables 5.30 & 5.33 only a two level multigrid algorithm is applied. This is very similar to findings in [76].

Furthermore, note that the $p = 3$ nonlinear h-multigrid results shown in Table 5.34 keep the same algorithmic properties as observed for $p = 2$. Note that for a W-cycle the nonlinear h-multigrid computation did not converge for the same solver settings as used for all computation in Table 5.34. A converged solution has been obtained when using a

lower initial CFL number but then no reduction in CPU time as compared to the single grid solver could be observed. An explanation why the h-multigrid W-cycle computation did not converge is an unfortunate CFL number control in this case. Since the early iterations are better in terms of residual reduction in comparison to the single grid and V-cycle computation the CFL number is chosen too large by the SER timestep control in a later step which causes instabilities. On the other hand the nonlinear p-multigrid algorithm with three levels shows very good results for both cycle types in terms of nonlinear fine level iterations and a similar behavior as the h-multigrid algorithms, with the same number of levels, shown in this section for the RANS-SA equations.

The main reduction in overall CPU time is achieved by applying the nonlinear multigrid algorithm in comparison to a single level solver, which cycle type to choose can differ for each test case and mesh. If all computations for single level, V-cycle, and W-cycle converge for one test case with all other solver settings fixed than a reduction in fine level Backward-Euler iterations can be always accomplished. In these cases the most fine level Backward-Euler iterations are preformed by the single level solver and the least with the nonlinear multigrid algorithm using a W-cycle. However, the fastest computation in terms of CPU time is always performed either the V-cycle or W-cycle nonlinear multigrid algorithm. The W-cycle is slower if the reduction in the fine level Backward-Euler iterations is not large enough to compensate the additional work on the coarse levels in comparison to a V-cycle.

Coarse level solving

Tables 5.35 & 5.36 include an investigation in which more than one smoother step on the lower levels of the nonlinear multigrid algorithm are performed. Thus, the lower level residuals are reduced more than in the one step approach discussed before. Please note that the computations shown in Table 5.35 (rows 1 & 8) are the same as shown in Table 5.29 (rows 2 & 3). Also Table 5.36 (row 1) and Table 5.32 (row 2) coincide.

As shown in Tables 5.35 (rows 1,4,6) & 5.36 (rows 1 & 2) it is not beneficial to increase the number of coarsest level smoother steps only without increasing the number of smoother steps on the other levels as well. Increasing the number of smoother steps on all coarse levels, with an increasing number of smoother steps starting from the top level down to the coarsest level, seems to be beneficial in terms of the numbers of iterations and CPU time. The CPU time reduction is at most 0.06 in normalized CPU time for the presented results. By using a W-cycle a larger reduction in CPU time can be observed, see Table 5.35 (row 8), in comparison to all other computations shown in this table. If a W-cycle is chosen information is transferred more often between the levels as compared to a V-cycle. Hence, the approximation on lower levels is updated with information from the higher levels more often which seems to be beneficial. Therefore, before altering the number of smoother steps on a level a W-cycle should be the preferred choice. Moreover, if the

no.	startup	NMG	levels	CPU time	nonlinear cycles
1	10^{-6}	v (1 1 1)	$h(3)$	0.87	19, 31, 96
2	10^{-6}	v (2 2 1)	$h(3)$	0.83	19, 19, 85
3	10^{-6}	v (4 2 1)	$h(3)$	0.82	19, 18, 77
4	10^{-6}	v (5 1 1)	$h(3)$	0.98	19, 31, 98
5	10^{-6}	v (5 2 2)	$h(3)$	0.81	19, 19, 54
6	10^{-6}	v (10 1 1)	$h(3)$	∞	19, ∞, -
7	10^{-6}	v (10 5 2)	$h(3)$	NaN	19, 12, NaN
8	10^{-6}	w (1 1 1)	$h(3)$	0.76	19, 31, 77

Table 5.35: L1T2 $(p = 2)$ $k\omega$-computation on the unstructured CEN mesh: effect of the number of coarse level smoother steps within the nonlinear multigrid algorithm.

no.	startup	NMG	levels	CPU time	nonlinear cycles
1	10^{-6}	v (1 1 1)	$h(3)$	0.63	72, 263, 25
3	10^{-6}	v (10 1 1)	$h(3)$	NaN	72, NaN, -

Table 5.36: MDA 30P30N $(p = 2)$ SA-computation on a structured mesh with 134 912 elements: effect of the number of coarse level smoother steps within the nonlinear multigrid algorithm.

difference in the number of smoother steps is too big between the levels instabilities seem to occur, as seen in Tables 5.35 & 5.36. Therefore, only one Backward–Euler smoother step is applied in the nonlinear multigrid algorithm in the remainder of this work, unless stated otherwise.

Conclusion

Figure 5.31 displays three computations included in Table 5.34. The unmarked line represents a single grid computation, whereas the line marked with "+" represents a three level nonlinear h-multigrid algorithm applying a V-cycle. Furthermore, the line marked with "•" represents a three level nonlinear p-multigrid algorithm and applying also a V-cycle. All computations are performed for $p = 3$ with a one step startup strategy. The effect which can be observed is that nonlinear iterations for $p = 3$ on the CEN mesh can be saved when a nonlinear multigrid algorithm is applied. Recall, that between the plotted iterations for nonlinear multigrid computations (+,•) additional iterations are performed on the lower levels which are not included in the plot. However, these iterations cost less CPU time than the the top level smoother iterations, as shown in Section 4.9. The most important remark to Figure 5.31 and all nonlinear multigrid computations is that this algorithm is most beneficial at the beginning of the computation. Since the asymptotic convergence rates at the end of the solution process for all shown computations stay nearly the same and only minor improvements are shown for the nonlinear multigrid algorithm. This remark is true for all converged computations of the nonlinear multigrid algorithm in this work. Therefore, the nonlinear multigrid algorithm accelerates the computation most at

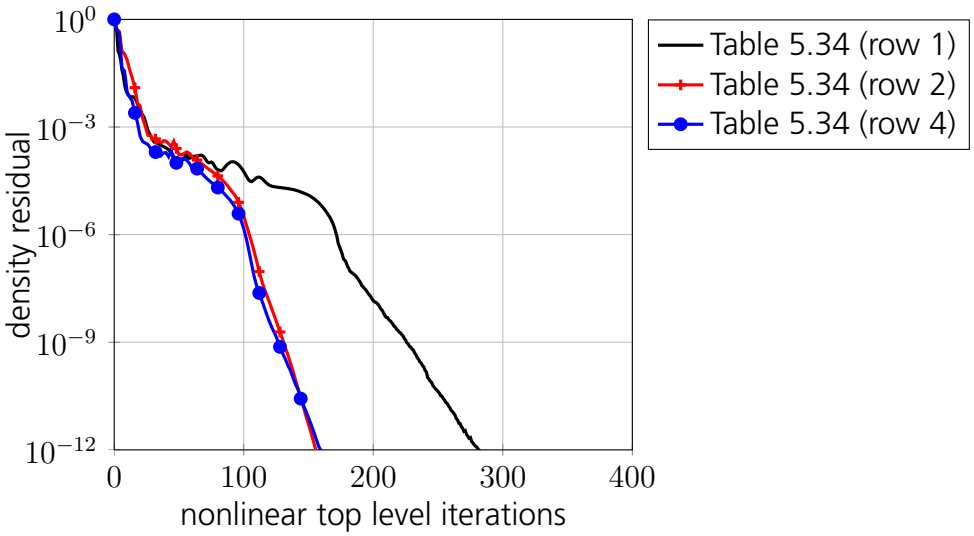

Figure 5.31: L1T2 $(p = 3)$ SA-computation on the CEN mesh, from Table 5.34: number of top level Backward–Euler iterations.

the beginning of the solution process. Hence, with the help of the nonlinear multigrid algorithm the regime of asymptotic convergence can be reached faster.

Solving the lowest level problems more accurately seems not as beneficial than changing the cycle type in terms of robustness and speed in comparison to applying one smoother step on all levels. Since one of the outcomes from Section 5.2.5 was that the linear multigrid applied as a preconditioner helps to enhance the solution process in the regime of asymptotic convergence, a combination which posses both characteristics is investigated in Section 5.2.8.

The best practice in terms of the nonlinear multigrid algorithm is to use only one smoother step every time the smoother is executed. The choice of the cycle type depends on the mesh and the test case, but most of the time a V-cycle is chosen. This is due to the fact that, in the majority of the test cases the reduction in nonlinear iterations while applying a W-cycle is not large enough to reduce the overall CPU time as well in comparison to a V-cycle computation.

5.2.8 Investigation of different solver choices

In this section, the main focus lies on the comparison of solver strategies available in the DG_ON framework. The baseline solver is the single level Backward–Euler algorithm with a Line–Jacobi preconditioned GMRes algorithm in combination with a startup strategy. All available solver within DG_ON will be compared to this baseline solver in terms of the number of nonlinear iterations and overall CPU time. Here all findings and best practice approaches from previous sections for each solver component are employed.

For all solvers the same solver settings are used unless it is stated otherwise. This includes the same number of GMRes iterations and the same initial CFL numbers on each level, for all computations shown within one table or figure.

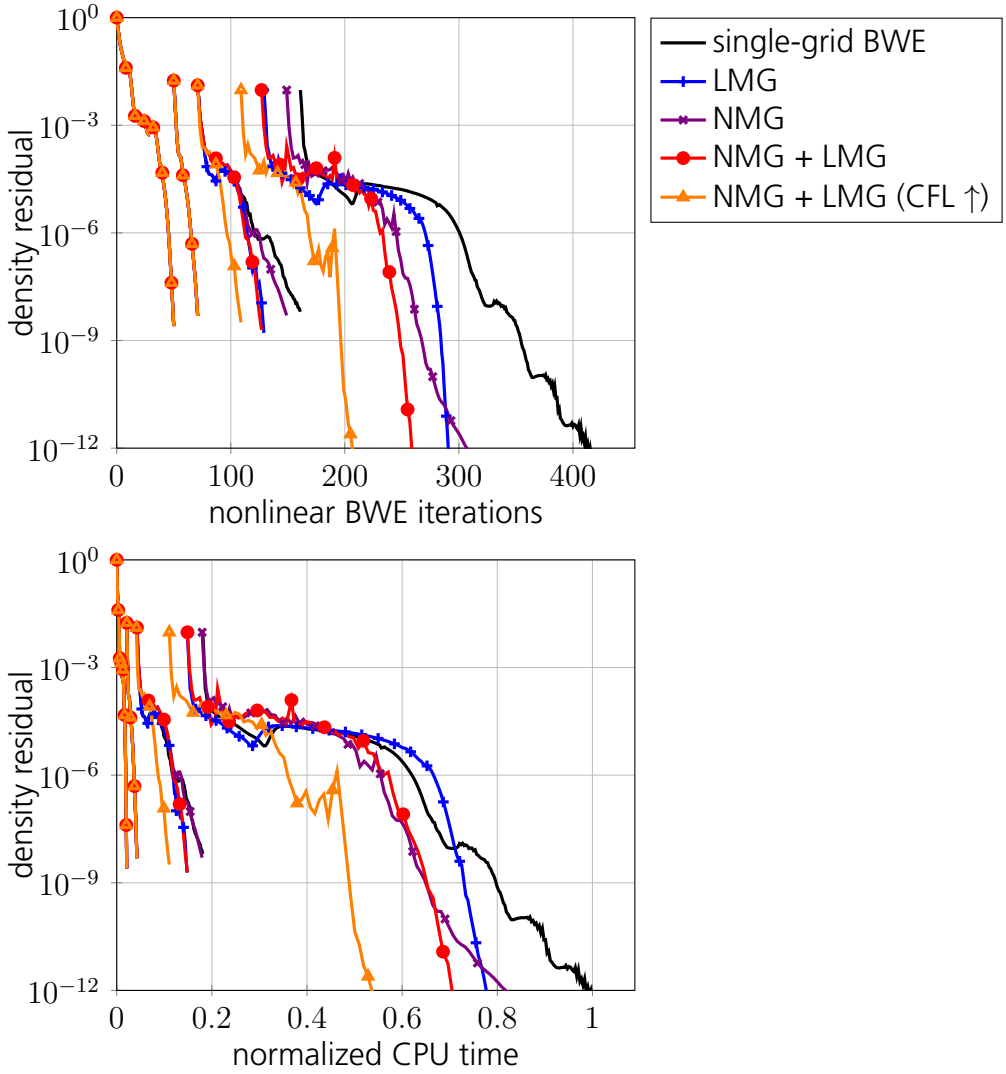

Figure 5.32: Nonlinear convergence for a L1T2 $k\omega$-computation at $p = 2$ on the CEN mesh for different combinations of solver algorithms with an h-startup strategy.

Figure 5.32 shows RANS-$k\omega$ computations of the L1T2 test case for $p = 2$ on the CEN mesh in combination with a 3 level h-startup strategy. The solver settings are chosen in

order to get good results for the single-grid computation. Both plots show the typical characteristics for each solver type, which have been stated before. The linear multigrid algorithm is particularly effective in the later part of asymptotic convergence on each level, whereas the nonlinear multigrid algorithm helps to improve in particular the initial phase of the convergence on each level. To confirm this statement the unmarked black line should be compared to the blue line marked with a "$+$" and the violet line marked with a "$\times$". The combination of a linear and nonlinear multigrid algorithm joins these beneficial characteristics and is superior to all other algorithms shown in Figure 5.32 in terms of nonlinear iterations and normalized CPU time for the same solver settings, see the red line marked with a "$\bullet$". Note that, the combination of a linear and nonlinear multigrid algorithm enters the regime of asymptotic convergence at the same point as the nonlinear multigrid algorithm but also has the same convergence rate as the linear multigrid at the end of the convergence process.

Moreover, if the initial CFL number on every level is increased applying the best settings for the linear and nonlinear multigrid combination then an additional reduction in CPU time and nonlinear iterations can be shown, see the orange lines marked with a "▲". Furthermore, if these higher initial CFL numbers are applied for all the order solver strategies shown in Figure 5.32 these algorithms did not converge. Thus, this shows a superior robustness of the nonlinear and linear multigrid combination. All computations shown, except the improved CLF number computation marked with a "▲", are also collected in the Table 5.37 (rows 1,2,3,4). In addition to that, p-multigrid computations with the same solver settings are included. The results for higher initial CFL numbers for the nonlinear and linear multigrid combination, both for the h- and p-version are collected in Table 5.38.

Since $p = 2$ results are computed all p-computations are 2 level approaches. At first a difference in terms of the startup strategy can be seen since the difference in the number of nonlinear iterations on the top level of the startup strategy in the computations of row 1 and row 5 results from the origin of the initial solution for the top level. This is due to the fact that all other solver settings are the same, including the timestep control. The computation where the initial solution is transferred from a lower polynomial degree on the same mesh is inferior to the initial solution transferred from a coarse mesh for the same polynomial degree, since 12 additional Backward–Euler iterations had to be performed on the top level of the p-startup strategy. This characteristic is typical for the computations shown in this work and aligns with findings in Section 5.2.2. However, which startup strategy to choose is often mesh dependent since the CPU time is the driving factor and an h-startup strategy especially on coarse grids may be too expensive in comparison to a p-startup strategy.

For the nonlinear p-multigrid algorithm only a reduction in nonlinear iterations can be observed, but this reduction it too small to also reduce the CPU time of the computation

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	10^{-8}	-	-	$h(3)$	1.00	21, 90, 255
2	10^{-8}	-	/	$h(3)$	0.79	21, 58, 167
3	10^{-8}	v	-	$h(3)$	0.88	21, 78, 174
4	10^{-8}	v	/	$h(3)$	0.73	21, 56, 136
5	10^{-8}	-	-	$p(2)$	1.17	279, 267
6	10^{-8}	-	/	$p(2)$	0.97	279, 179
7	10^{-8}	v	-	$p(2)$	1.25	279, 260
8	10^{-8}	v	/	$p(2)$	1.02	279, 174

Table 5.37: L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: solver comparison with same settings, see Figure 5.32.

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	10^{-8}	v	/	$h(3)$	0.57	21, 48 104
2	10^{-8}	v	/	$p(2)$	0.70	279, 88

Table 5.38: L1T2 $(p = 2)$ $k\omega$-computation on the CEN mesh: higher CFL numbers in comparison to Table 5.37.

in comparison to the associated single-grid computation given in row 5 [76]. This might be due to the fact that only 2 levels can be exploited. However, the linear and nonlinear p-multigrid combination also joins both solver behaviors shown for the nonlinear and linear multigrid separately. This results in an overall reduction in the number of nonlinear iteration in comparison to all other p-algorithms approaches shown in Table 5.37. The overall CPU time is not smaller due to the nonlinear multigrid cost in comparison to the gained nonlinear iteration reduction. Overall it seems that the h-multigrid algorithms are more feasible, since the number of levels is not limited by the polynomial degree, and more applicable to all meshes, in particular to finer meshes. Moreover, if the initial CFL number on the top level is increased for the p-multigrid combination a significant CPU time reduction can be observed, see row 2 in Table 5.38. This demonstrates the beneficial properties of the nonlinear and linear multigrid combination even for p-computations.

Moreover, the p-multigrid algorithms are most of the time an improvement in comparison to single-grid computations and seem to show similar results as the h-multigrid algorithms if the level depth is the same for both algorithms. This is shown in Table 5.39 for computations on the same mesh but for the SA-turbulence model and a polynomial degree of $p = 3$.

In this test case the nonlinear p- and h-multigrid algorithm seem to produce similar results and both enhance the convergence process not only in terms of nonlinear iterations but also in CPU time. Moreover, since a one step startup strategy is used no difference in nonlinear iterations on the top level of the startup strategy for the single-grid computa-

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	1	-	-	$p(3)$	1.00	1, 1, 366
2	1	-	v	$p(3)$	0.90	1, 1, 169
3	1	-	/	$p(3)$	0.78	1, 1, 174
4	1	v	-	$p(3)$	0.70	1, 1, 162
5	1	v	v	$p(3)$	0.70	1, 1, 110
6	1	v	/	$p(3)$	0.61	1, 1, 112
7	1	-	-	$h(3)$	1.02	1, 1, 365
8	1	-	v	$h(3)$	0.87	1, 1, 168
9	1	-	/	$h(3)$	0.72	1, 1, 170
10	1	v	-	$h(3)$	0.61	1, 1, 144
11	1	v	v	$h(3)$	0.61	1, 1, 102
12	1	v	/	$h(3)$	0.48	1, 1, 96

Table 5.39: L1T2 $(p = 3)$ SA-computation on the CEN mesh: solver comparison with same settings.

tions in rows 1 & 7 of Table 5.39 can be seen. Furthermore, in Table 5.39 (rows 5 & 11) a V-cycle is applied in the linear multigrid algorithm and no reduction in computational time in comparison to the computations in Table 5.39 (rows 4 & 10) can be shown. Since the reduction in the number of nonlinear iterations does not compensate for the additional effort of the V-cycles applied in the linear multigrid preconditioner. This is similar to the findings from Table 5.37 where the linear p-multigrid algorithm in combination with the nonlinear p-multigrid algorithm performed worse in terms of CPU time than without the nonlinear multigrid algorithm.

The linear and nonlinear multigrid combinations, Table 5.39 (rows 6 & 12), are again the fastest algorithms while applying the same solver settings. In addition to that the h-multigrid algorithms are again faster in terms of normalized CPU time, as seen in Table 5.37. Moreover, the same solver characteristic as described in previous sections and Table 5.37 also hold for the computations shown in Table 5.39.

In Tables 5.40 - 5.43 computations of the nonlinear and linear multigrid combinations in comparison to single-grid computations are shown for different test cases. All computations in these tables show that the nonlinear iterations are reduced if a nonlinear and linear multigrid combination is applied. Table 5.40 reinforces the fact that a h-startup strategy is more robust than the p-counter part. Moreover, if the solver settings are altered such that the number of GMRes iterations is increased and the initial CLF number on the top level is reduced the p-computations where the multigrid combination is used converges as well. This computation has then a normalized CPU time of 1.55 in comparison to the h-startup single-grid computation. Furthermore, the p-startup single-grid computation does not converge with the same solver settings as used for the converged

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	-	$h(3)$	1.00	35, 89, 311
2	10^{-6}	v	v	$h(3)$	0.61	35, 52, 70
3	10^{-6}	-	-	$p(2)$	∞	279, ∞
4	10^{-6}	v	v	$p(2)$	NaN	279, NaN

Table 5.40: MDA 30P30N $(p = 2)$ $k\omega$-computation on the 33 728 element mesh: solver comparison with same settings.

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	-	$h(3)$	1.00	75, 959, 201
2	10^{-6}	v	v	$h(3)$	0.66	75, 526, 52

Table 5.41: MDA 30P30N $(p = 2)$ SA-computation on the 134 912 element mesh: solver comparison with same settings.

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	-	$p(2)$	1.00	84, 77
2	10^{-6}	v	/	$p(2)$	0.87	84, 60
1	10^{-6}	-	-	$h(2)$	1.17	28, 78
2	10^{-6}	v	/	$h(2)$	0.80	28, 30

Table 5.42: VFE-2 $(p = 2)$ $k\omega$-computation on the 110 528 element mesh: solver comparison with same settings.

no.	startup	NMG	LMG	levels	CPU time	nonlinear iter.
1	10^{-6}	-	-	$p(2)$	NaN	162, NaN
2	10^{-6}	v	/	$p(2)$	1.00	162, 104
3	10^{-6}	-	-	$h(3)$	∞	64, 39, ∞
4	10^{-6}	v	/	$h(3)$	0.83	64, 24, 36

Table 5.43: NASA Trap Wing $(p = 2)$ SA-computation on the 744 704 element mesh: solver comparison with same settings.

p-multigrid combination. Thus, this supports the previous findings about the multigrid combination.

Results for the VFE-2 test case are shown in Table 5.42. They demonstrate that also in 3D the multigrid combination is superior in comparison to the single-grid computations. Moreover, the multigrid combination also joins both beneficial solver characteristics also for 3D computations. The inferior run time for the h-single-grid computation in comparison to the p-single-grid computations is due to the fact that the h-startup strategy takes 0.41 in normalized CPU time, whereas the p-startup strategy only takes 0.24.

In Table 5.43 the best solver settings for the p-multigrid combination are applied to the NASA Trap Wing test case. Both single-grid computations do not converge with these

solver settings, thus, showing the robustness of the multigrid combination. Furthermore, the h-multigrid combination is superior in terms of CPU time and nonlinear iterations while applying the same solver settings. When the initial CFL number on the top level of the startup strategies is halved both single-grid computations will converge but with a inferior overall CPU time, the h-startup strategy single-grid computation then takes 1.53 in normalized CPU time and the p-startup strategy takes 2.05.

Conclusion

Following from the presented results the "best practice" approach will always be the combination of the nonlinear and linear multigrid algorithm, where the linear multigrid algorithm is used as a preconditioner for the GMRes method. This combination joins all positive characteristics of both algorithms. In fact, it is more robust since a smaller maximal number of GMRes steps and a larger CFL number choice is possible than for all other algorithms considered. Moreover, since an h-startup strategy is always favorable, in particular on fine meshes, an h-multigrid combination will be the standard approach. Furthermore, the additional CPU time per iteration in comparison to all other algorithms is in most cases compensated by an overall better convergence behavior. Beyond that the overall CPU time is always reduced in comparison to single-grid computations when both run at their individually best solver settings.

In some rare cases if the solver settings are chosen such that the e.g. the linear multigrid algorithm is not required for an optimal convergence rate this algorithm will only cost additional time and will not reduce the overall CPU time. However, in all cases solver settings can be found that will reduce the overall CPU time in comparison to the computation described above. The same is true for the nonlinear multigrid algorithm. Moreover, if in a comparison with the same solver settings the settings are chosen very unfortunate for one of the multigrid algorithms the combination will always profit from the beneficial properties of the other algorithm to save CPU time. Up to this point for all test cases no best setting single-grid computation was faster than a best setting multigrid computation within the DG_ON framework.

5.2.9 h- and p-independency

In this section the independence of the proposed solver algorithms from the mesh size and the polynomial degree is investigated. The focus of this investigation lies on the nonlinear and linear multigrid algorithm combinations for different mesh hierarchies and polynomial degrees. For industrial near test cases, in particular for turbulent test cases and the associated meshes, theoretical order assumptions and solver independence for grid size or polynomial degree often do not hold for modern CFD codes [41]. The multigrid combination is characterized as a nonlinear multigrid algorithm with the Backward-Euler smoother, where in every linear solutions process a linear multigrid preconditioned GMRes method is used.

p-independence of the h-multigrid combination

When the polynomial degree of the underlying discretization is increased the number of DoF also increase but the computational mesh stays the same. Increasing the polynomial degree helps to better resolve underresolved flow regions. While changing the polynomial degree, an h-multigrid algorithm could produce a similar convergence history for all polynomial degrees if the same solver settings are applied. The smoother in the linear h-multigrid algorithm is the Line–Jacobi algorithm and the lines nearly stay the same for all polynomial degrees. Therefore, the h-multigrid preconditioner stays the same from a mesh point of view but the size of the matrix blocks employed per mesh element within a line grows with the polynomial degree. This implies, that the outer algorithms do not see a difference due to the change in the polynomial degree but the inverted Jacobi–blocks within the Line–Jacobi algorithm get larger although the number of blocks and, therefore, the matrix structure stays the same for every polynomial degree. Hence, the convergence history should stay nearly the same but the effort in terms of CPU time should grow with increasing the polynomial degree.

In this paragraph the L1T2 test case on the unstructured CEN mesh is computed with a 3 level h-multigrid combination and a one step startup strategy. In Figure 5.33 on the left exactly the same solver settings within the 3 level h-multigrid combination are applied. The only difference is the polynomial degree of the computed solution. This result indicates that on this mesh and for this test case a p-independency of the h-multigrid combination is obtained since the solver algorithm takes nearly the same number of iterations. Moreover, since the convergence rates at the end of the solution process stay the same the linear problems are solved to a comparable linear residual in each iteration for each polynomial degree. This implies that, the inverted Jacobi–blocks which only differ in size for different polynomial degrees are computed with a comparable accuracy for each polynomial degree because all other algorithms applied stay exactly the same in all cases. This confirms the theoretical assumptions stated above about the linear h-multigrid preconditioner.

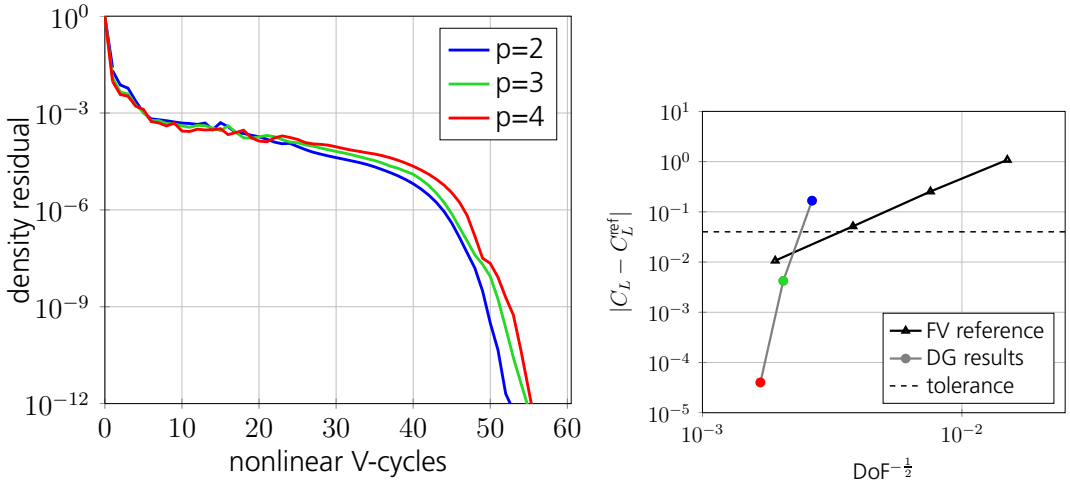

Figure 5.33: Comparison between RANS-SA DG_ON results for $p = 2$ (blue), $p = 3$ (green) and $p = 4$ (red) on the CEN mesh in terms of the convergence history of the density residual component on the left and the error in the lift coefficient in comparison to reference computations on the right.

The theoretical assumptions that the effort in terms of CPU time should grow with increasing the polynomial degree also holds. For the same solver settings and convergence history, if the CPU time is normalized with the computational time of the $p = 2$ computation, the $p = 3$ computation takes 2.99 in normalized CPU time and the $p = 4$ computation takes 8.24.

h-independence of the h-multigrid combination

A brief investigation will be presented since it was shown in Sections 4.8 & 4.9 that the computational cost of integration on lower levels stays the same for all levels during a startup strategy. Moreover, it was shown that the additional cost by adding a level to a nonlinear multigrid cycle may not scale optimally with the coarsening rate. Therefore, theoretical convergence rate of $\mathcal{O}(N)$ of the CPU time in correlation to the grid size will not hold for the h-multigrid combination.

The investigation will be performed on the last three meshes of the MDA 30P30N nested mesh hierarchy with 8 432 up to 134 912 elements. On this mesh sequence $p = 2$ results are shown for the RANS-SA equations. The algorithm applied in Figure 5.34 is the nonlinear and linear h-multigrid combination with a h-startup strategy with a residual tolerance of 10^{-6}. All computations use the same solver settings, like the number of maximal GMRes iterations and CFL number control. The only difference is in the number of levels.

On the mesh with 8 432 elements a single-grid computation is performed, whereas on the mesh with 33 728 elements a two level h-multigrid combination is used and on the mesh with 134 912 elements a three level h-multigrid combination. Moreover, a coarsening rate $c_{\text{rt}} = 4$ is used. Hence, the coarse level of all computations in Figure 5.34 has approximately 8 000 elements. Therefore, the results shown indicate a scaling over

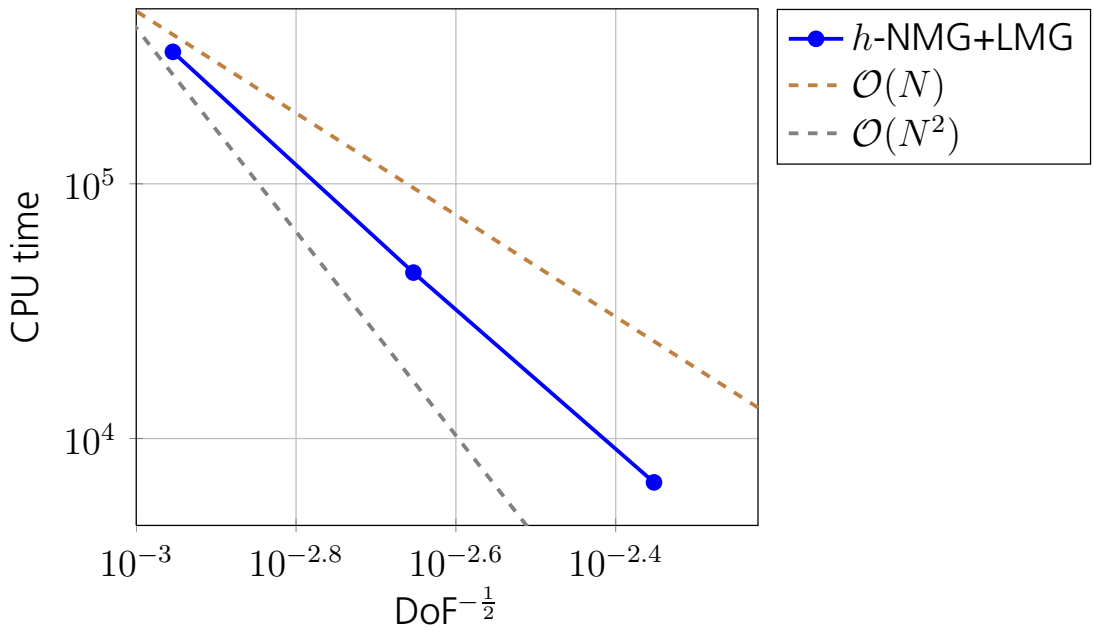

Figure 5.34: Convergence rates of RANS-SA computations of DG_ON result for $p = 2$ on the MDA 30P30N mesh hierarchy.

the mesh sequence between $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$. This result is expected since it is shown in Section 4.9 that additional levels in the multigrid algorithms produce additional cost but this cost is limited and scales with the problem size of the coarse level if a Galerkin transfer is used. This results in additional CPU time for each additional level. Moreover, the startup strategy, with a residual tolerance of 10^{-6} on each coarse level, takes more CPU time. Following from Section 4.8 the time spend for assembling the Jacobian on the 8 000 element level in the startup strategy will increase with the fine mesh size, while the cost for algebra operations on this level will stay the same for all shown computations. Note, that his additional cost for assembling the Jacobian on lower levels is not present on the 8 000 element level of the h-multigrid combination since a Galerkin transfer is used. Overall this result aligns with findings of other authors for turbulent test cases [41].

Conclusion

The theoretical assumptions regarding the h-multigrid combination are confirmed in this section for two turbulent high-lift test cases. The independence in the convergence characteristics over the number of nonlinear iterations for different polynomial degrees is shown for a unstructured mesh of the L1T2 test case. This independence could rise from the fact that the solver settings for the linear problems are sufficient even for $p = 4$ and therefore, solved similar for every polynomial degree resulting in a comparable convergence rate at the end of the computations. Moreover, the only difference for all polynomial degrees within the proposed algorithms is the size of the Jacobian blocks and their inversion in the Line–Jacobi method, **Algorithm 4** A convergence rate between $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ of the CPU time in correlation to the grid size can be shown for the h-multigrid

combination on the MDA 30P30N test case. Moreover, this convergence rate aligns with findings from Sections 4.8 & 4.9 and other authors [41].

5.3 Framework benchmark

In this subsection test cases from Section 5.1 are revisited and results are compared with other codes. This comparison will be performed in terms of accuracy of the results and the computational cost. To this end, a work unit is introduced which should reflect the cost of a computation and provide fair comparrisons with other CFD codes.

Definition 5.1:

A work unit is defined to be the time TauBench [19] takes to run on the same machine as the CFD computations are performed, see Appendix B. It should be run at least three times to obtain an average wall clock time $T1$.

$T2$ is defined as the wall clock time taken by the CFD solver neglecting the initialization, post-processing data preparation time and file I/O time.

The computational cost of the CFD computation in terms of work unit is then given by

$$\frac{N1 \cdot T2}{T1},$$

where $N1$ is the number of processors used for the CFD computation.

Note that TauBench was used as a reference for work units at the 1st & 2nd International Workshop on High-Order CFD Methods (HOW I&II) and in the EC funded IDIHOM project [33], see Appendix B. Furthermore, all CFD solver comparisons in this section use data from one of these sources and the time comparison is given in work units as defined in Definition 5.1. Moreover, through the introduction of the factor $N1$ parallelization is considered in this benchmark. This is necessary because all of the results shown were performed on multiple CPU processors. For the cluster system on which the CFD computations for this work were perfromed thesis took place $T1 = 8.4\,s$ was obtained. Furthermore, for all results $T2$ was taken as the first time the density residual was reduced by more than 10 orders of magnitude in comparison to the freestream density residual. This measuring point for $T2$ was also used at HOW I&II and in the IDIHOM project.

All results of the DG_ON framework presented in this section where computed based on the nonlinear and linear multigrid combination investigated in Section 5.2.8. Recall that this algorithm combination is the fastest in terms of CPU time and in terms of robustness due to the wide range of possible solver settings.

L1T2 high-lift three element airfoil

In the Figures 5.35 & 5.36 every • represents a computation for a different polynomial degree on a given mesh. On the CEN mesh, results of the RANS-SA equations for $p = 1$ up to $p = 4$ are shown, whereas for the WUT mesh results for $p = 2$ and $p = 3$ are shown.

The lines marked with a ▲ are results from the unstructured node-centered FV TAU code [41] for the SA turbulence model on a mesh hierarchy of four structured meshes.

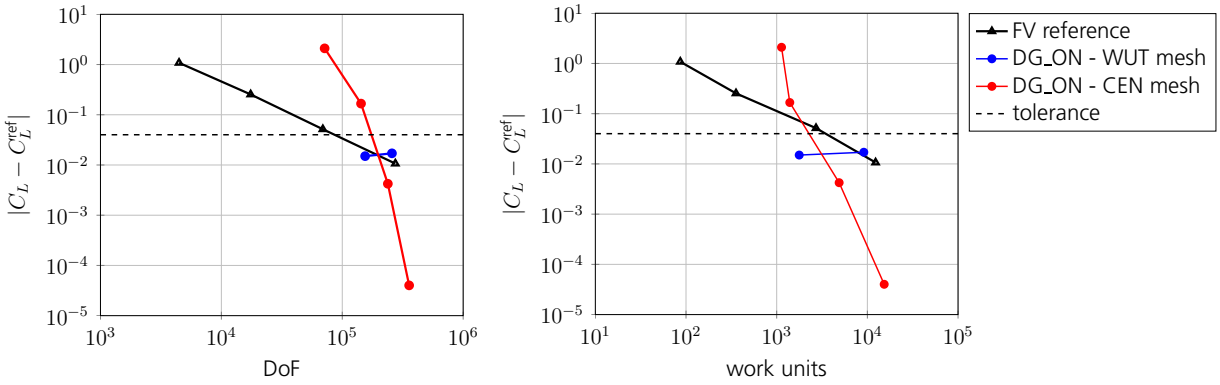

Figure 5.35: Mesh convergence of the error in lift over DoF (left) and work units (right) for reference computations and DG results of the L1T2 test case.

For the lift coefficient the lines representing the accuracy per work unit of DG_ON and the TAU code are similar to the lines representing the accuracy per DoF, but the points at which the lines intersect are shifted. Hence, DG_ON computes results faster for the same number of DoF as the TAU code. The reduction in cost is different for each polynomial degree. For $p = 1$ to 3 the reduction in cost is clearly visible in Figure 5.35 The cost per DoF for the TAU computation on the finest grid is 0.0448 and for the $p = 4$ computation of DG_ON it is 0.429. Moreover, the cost per DoF for the $p = 2$ computation on the CEN mesh with DG_ON is 0.0097. Depending on the polynomial degree the DG results are even more accurate for C_L, for the comparable number of DoF. The corresponding plots for the error in the drag coefficient are shown in Figure 5.36.

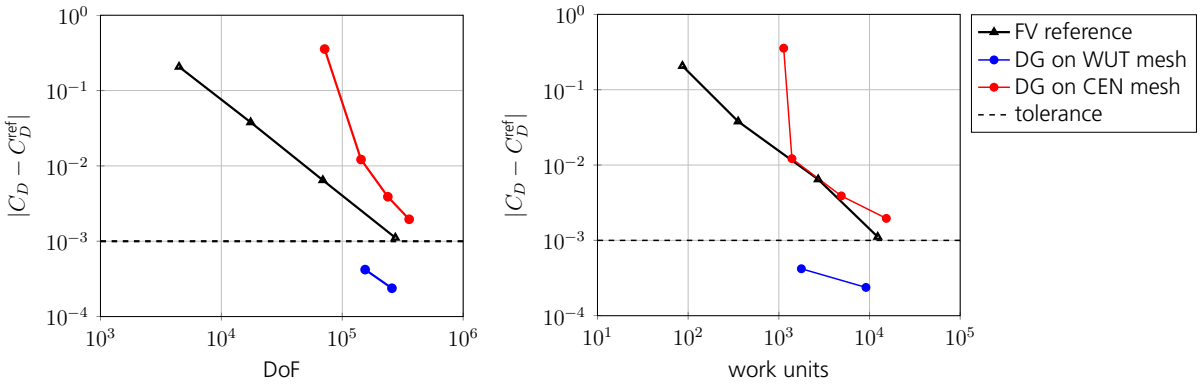

Figure 5.36: Mesh convergence of error in drag over DoF (left) and work units (right) for reference computations and DG results of the L1T2 test case.

For the drag coefficient the results are similar to the results for the lift coefficient. The results on the CEN mesh for the drag coefficient are not satisfying in terms of accuracy due to the poor mesh resolution in some areas, as mentioned before in Section 5.1.1. The

results on the WUT mesh for $p = 2$ and $p = 3$ for the drag coefficients show the same characteristics as seen for the lift coefficient, even in terms of cost per accuracy.

The conclusion drawn from this test case and the comparisons shown is that in order to achieve a reduction in work units and an increased accuracy in C_L and C_D over DoF for the given tolerance a discretization of polynomial degree of $p = 2$ or $p = 3$ should be used. This aligns with findings published in [77], which stated that in this area of application the highest gain in terms of computational effort per accuracy is obtained by approaches up to third or fourth order.

MDA 30P30N three element airfoil

In the following, the MDA 30P30N test case will be computed based on $p = 2$ discretizations on a hierarchy of meshes. Computations will be shown four higher order quadrangular meshes from 2 108 elements up to 134 912 elements. Every "•" in Figures 5.37 represents a solution to the RANS-SA equations on one mesh of this hierarchy.

The $p = 2$ RANS-SA results are compared again against SA-results from the TAU code, see Figure 5.37. The TAU results were computed on three different triangular meshes, with 119 510, 240 955 and 485 832 elements.

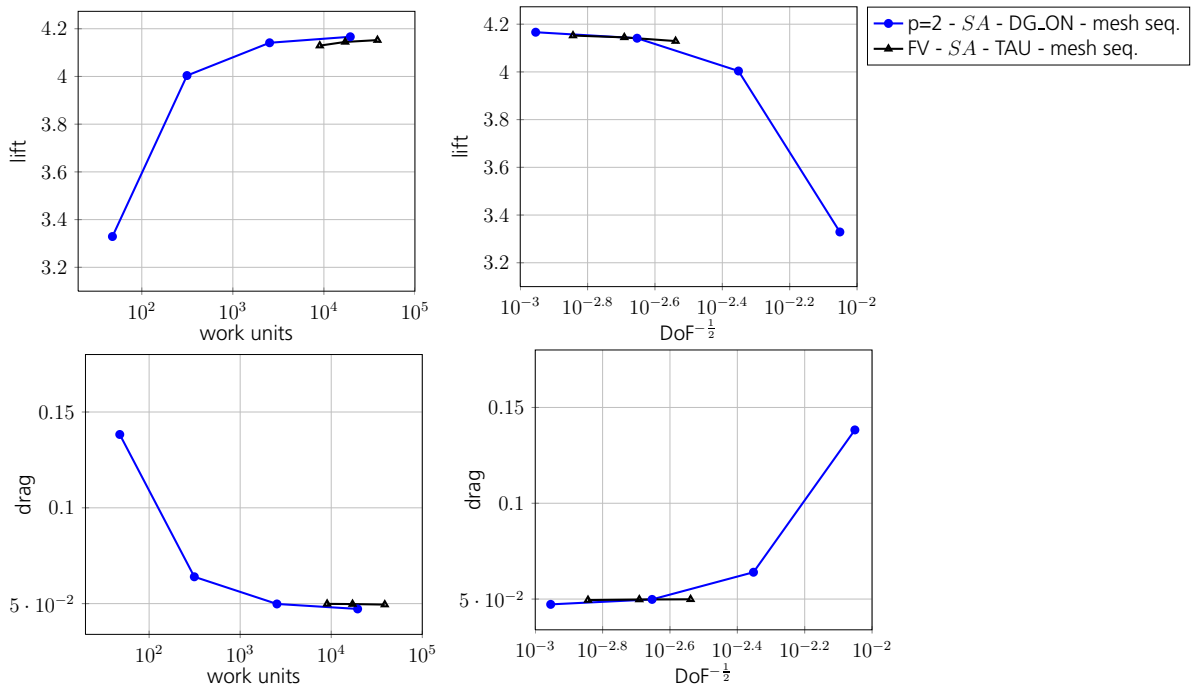

Figure 5.37: Results of the MDA 30P30N testcase in combination with the RANS-SA equations. Left: Lift and drag in correlation to work units. Right: Lift and drag in correlation to DoF.

The DG_ON computation on the mesh with 33 728 elements, third mesh in the hierarchy, gives satisfying results in terms of lift and drag coefficients. The "•" representing this computation lies in between the results form the first and second mesh of the TAU mesh hierarchy in terms of DoF, as seen in the right plots of Figure 5.37. However, this compu-

tation takes less work units than all TAU computations, as seen in the left plots of Figure 5.37. Moreover, a lower computational cost per DoF can be seen for every DG_ON computation in Figure 5.37 in comparison to the TAU results. Hence, the findings from the L1T2 test case also hold for a structured mesh hierarchy for the same polynomial degree.

The DG_ON RANS-$k\omega$ computations are marked with a "$\times$" for $p = 1$ and $p = 2$, in Figure 5.38. Note, that the $p = 2$ result on the finest mesh is not shown here because the density residual could be reduced by 7 orders of magnitude only. Therefore, this computation fullfills not the work unit assessment criterion which was mentioned above. The DG_ON computations are compared to DG results provided by the University of Bergamo and their CFD code Migale on a structured 12 872 element mesh for a $p = 2$ and $p = 3$ discretization while applying the $k\omega$-turbulence model, represented by the lines marked with a "$+$" in Figure 5.38. The Migale code also uses a strongly implicit solver approach.

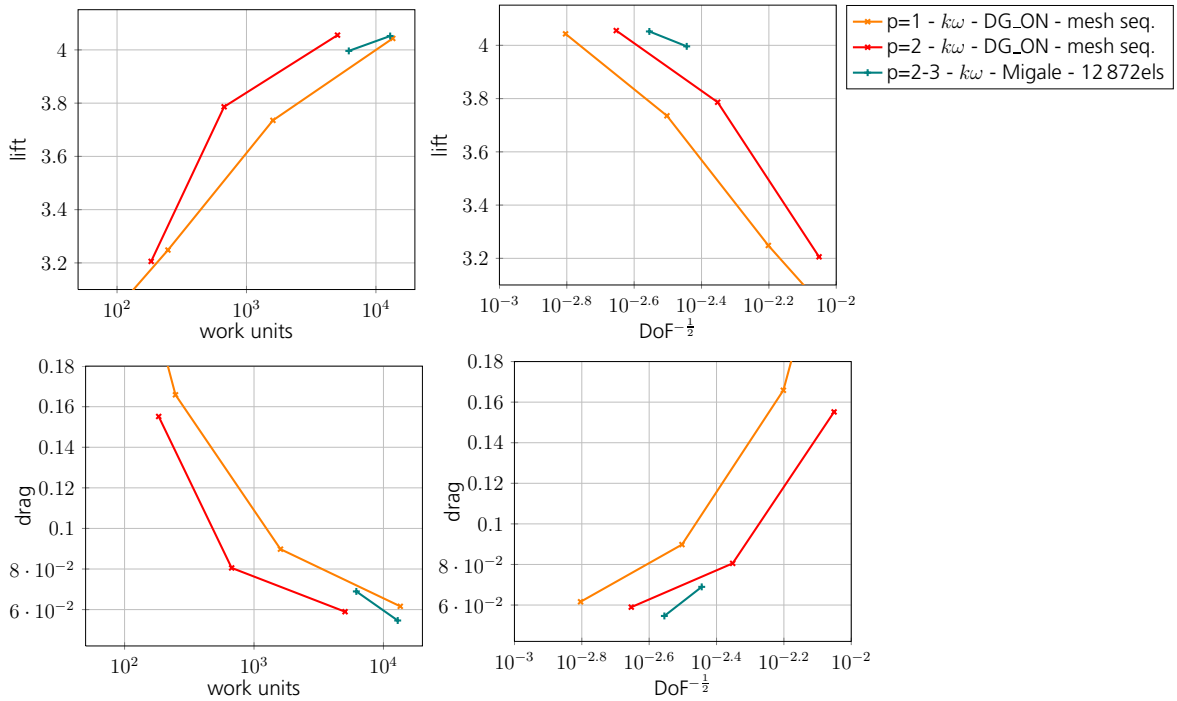

Figure 5.38: Results of the MDA 30P30N testcase in combination with the RANS-$k\omega$ equations. Left: Lift and drag in correlation to work units. Right: Lift and drag in correlation to DoF.

The same conclusions drawn for the RANS-SA equations in comparison with the TAU code can be drawn here as well. A lower computational cost per DoF in comparison to the Migale code can be seen for every DG_ON computation in this test case. The accuracy of the RANS-$k\omega$ computation on the mesh with 33 728 elements, i.e. the last red "$\times$" in all plots of Figure 5.38, is similar to that of the $p = 3$ RANS-$k\omega$ computation on the mesh with 12 872 elements from the University of Bergamo. Furthermore, the computations of DG_ON are faster although they include more DoF.

NASA Trap Wing

For the NASA Trap Wing no data in terms of work units is available for comparisons. The lift coefficient accuracy is shown in Figure 5.39 over DoF and work units.

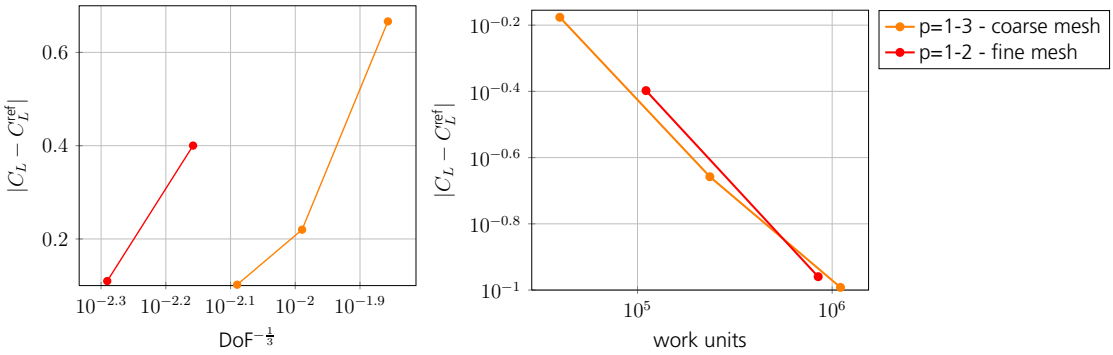

Figure 5.39: Mesh and order convergence of the error in the lift coefficient over DoF (left) and work units (right) of the NASA Trap Wing test case.

The error of the lift coefficient on these meshes in comparison to experimental data is pretty high after all. However, on both meshes the error gets smaller as the polynomial degree is increased. Also the error decreases as the mesh is refined for a fixed polynomial degree.

6 Conclusions

Summary

Based on the DG discretization of the Reynolds-Averaged Navier-Stokes (RANS) equations on unstructured meshes using non-parametric basis functions a common framework for the construction of coarse level problems for convergence acceleration has been developed and analyzed. The coarse level problems are formed by agglomeration of the underlying mesh or by a reduction of the associated polynomial degree has been analyzed in this work.

These coarse level problems can be exploited in a number of ways to enhance solver robustness and efficiency, including mesh and order sequencing and linear as well as nonlinear multigrid variants. The resulting algorithms can be expressed in the same way for p- and h-multigrid algorithm formulations and differ only in the concrete form of the transfer operators. These algorithms were used for the construction of robust, efficient and scalable solver algorithms for a higher-order discontinuous Galerkin (DG) discretization of steady-state turbulent flow problems in aerodynamic applications. Two turbulence models where applied to several test cases in 2D and 3D which are well known and investigated in the field of Computational Fluid Dynamics (CFD). Moreover, structured and unstructured meshes where applied. Furthermore, a comparison to different state of the art DG results is shown within this investigation demonstrating the capability of the developed strategies and a "best practice" approach is introduced.

Main algorithmic findings

The different ways of exploiting coarse level problems mentioned above are motivated by different aspects of typical solver behavior. Mesh and order sequencing, which correspond to the use of a full multigrid (FMG) approach if a multigrid algorithm is employed, help to improve the initial solution on each level, which is particularly beneficial in the case of an implicit relaxation scheme.

An h-startup strategy should always be preferred for medium and fine grids over a p-startup strategy since is seems to be more robust and helps to accelerate the solution process on the finest level. Moreover, a difference in the startup strategy for the investigated turbulence models is found. For the RANS-$k\omega$ equations a startup strategy with a residual

tolerance of 10^{-6} on each coarse level should be employed, whereas for the RANS-SA equations such low residual tolerance is necessary for higher polynomial degrees or finer meshes only.

The nonlinear multigrid algorithm accelerates the solution process in pseudo-time in comparison to a single-grid solver. Using a strong implicit smoother helps to improve in particular the initial phase of the convergence on each level. Furthermore, applying a W-cycle while utilizing one smoother step seem to be beneficial in terms of reducing the number of nonlinear iterations compared to a V-cycle, but an associated reduction in overall CPU time is mostly not visible due to the increased cost of the cycle.

The linear multigrid algorithm used as a preconditioner within the GMRes method helps to alleviate the stiffness due to high order and fine meshes encountered in the linear systems arising from the implicit operator. It is particularly effective in the later part of asymptotic convergence on each level, in which the stiffness is further increased due to the use of larger CFL numbers and thus less stabilization. Moreover, applying more smoother steps on the lower levels and using stronger cycles is always beneficial in terms of robustness and convergence rates. However, the linear multigrid algorithm used as linear solver seems to be much slower and less robust then the linear multigrid preconditioned GMRes method. It was shown that, the linear multigrid preconditioned GMRes method is the best solution strategy in terms of linear solving procedures.

The beneficial properties described above of the nonlinear and linear multigrid algorithm can be utilized in a combination of both algorithms. This combination has been demonstrated as the most robust and efficient solver algorithm. The h-multigrid combination is favorable over the p-multigrid combination, since it is more robust for all test cases and in most cases more beneficial in terms of CPU time. This finding aligns also with the findings for the investigated h- or p-startup strategies, where a h-startup strategies is favorable.

The "best practice" nonlinear and linear multigrid combination is compared to other CFD codes in terms of work units. The results indicate that the cost per DoF measured in work units is superior in comparison to results published from other CFD codes. Following from this, the multigrid combination fulfills all desired characteristics which were formulated at the beginning of this work in order to solve steady-state turbulent flow problems and is characterized as the "best practice" solver.

In detail, the "best practice" solver is a nonlinear h-multigrid algorithm with a Backward–Euler smoother. A V-cycle is applied with one smoother step on every level even on the coarsest level. The linear problems on every level are solved by a h-multigrid preconditioned GMRes method. One V-cycle with four smoother steps on all levels except the coarsest level is performed every time the preconditioner is executed. On the coarsest level of the linear multigrid algorithm 12 smoother steps are performed. The number of GMRes iterations varies per test case between 20 and 50 maximal iterations with a max-

imal residual reduction of 6 orders of magnitude. An h-startup strategy is chosen with a residual tolerance of 10^{-6}. Moreover this h-startup strategy always performs a p–startup strategy on the coarsest grid inorder to accelerate the solution process further.

For this "best practice" solver p-independence on a unstructured mesh is indicated. In case of the h-independence an optimal behavior of $\mathcal{O}(N)$ could not be archived, which most likely is due to the fact that the startup computations on the coarser grids do not scale in terms of the integration effort since it is depending on the top level mesh size as shown in Sections 4.8 & 4.9. This non-optimal behavior was observed by other authors for turbulent test cases as well [41].

Outlook

Finally, some topics are mentioned as a perspective for further research. An interesting opportunity is the combination of an h- and p-multigrid approach, which is not yet implemented in the current framework and requires some investigation on favorable strategies concerning the order in which mesh resolution and polynomial degree are reduced. Moreover, applying DG_ON to turbulent transonic flow conditions requires some further investigation with respect to the shock capturing in combination with multigrid algorithms and seems to be an interesting topic for further research.

A Convective flux Jacobian

The Jacobian of the normal convective flux in 3D is defined as:

$$\mathcal{F}_n^{c\prime} := \frac{d(\mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n})}{d\mathbf{u}} = \sum_{j=1}^{3} n_j \cdot F_j, \quad j = 1, 2, 3,$$

where for the RANS-$k\omega$ equations the fluxes are given by

$F_1 =$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -v_1^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (3-\gamma)v_1 & (1-\gamma)v_2 & (1-\gamma)v_3 & \gamma-1 & \frac{5}{3}-\gamma & 0 \\ -v_1 v_2 & v_2 & v_1 & 0 & 0 & 0 & 0 \\ -v_1 v_3 & v_3 & 0 & v_1 & 0 & 0 & 0 \\ (\gamma-1)v_1\mathbf{v}^2 - \gamma v_1 E + (\gamma - \frac{5}{3})v_1 k & \gamma E - \frac{\gamma-1}{2}(2v_1^2 + \mathbf{v}^2) + (\frac{5}{3}-\gamma)k & (1-\gamma)v_1 v_2 & (1-\gamma)v_1 v_3 & \gamma v_1 & \left(\frac{5}{3}-\gamma\right)v_1 & 0 \\ -v_1 k & k & 0 & 0 & 0 & v_1 & 0 \\ -v_1 \omega & \omega & 0 & 0 & 0 & 0 & v_1 \end{pmatrix},$$

$F_2 =$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -v_1 v_2 & v_2 & v_1 & 0 & 0 & 0 & 0 \\ -v_2^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (1-\gamma)v_1 & (3-\gamma)v_2 & (1-\gamma)v_3 & \gamma-1 & \frac{5}{3}-\gamma & 0 \\ -v_2 v_3 & 0 & v_3 & v_2 & 0 & 0 & 0 \\ (\gamma-1)v_2\mathbf{v}^2 - \gamma v_2 E + \left(\gamma - \frac{5}{3}\right)v_2 k & (1-\gamma)v_1 v_2 & \gamma E - \frac{\gamma-1}{2}(2v_2^2 + \mathbf{v}^2) + \left(\frac{5}{3}-\gamma\right)k & (1-\gamma)v_2 v_3 & \gamma v_2 & \left(\frac{5}{3}-\gamma\right)v_2 & 0 \\ -v_2 k & 0 & k & 0 & 0 & v_2 & 0 \\ -v_2 \omega & 0 & \omega & 0 & 0 & 0 & v_2 \end{pmatrix},$$

$F_3 =$

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -v_1 v_3 & v_3 & 0 & v_1 & 0 & 0 & 0 \\ -v_2 v_3 & 0 & v_3 & v_2 & 0 & 0 & 0 \\ -v_3^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (1-\gamma)v_1 & (1-\gamma)v_2 & (3-\gamma)v_3 & \gamma-1 & \frac{5}{3}-\gamma & 0 \\ (\gamma-1)v_3\mathbf{v}^2 - \gamma v_3 E + \left(\gamma - \frac{5}{3}\right)v_3 k & (1-\gamma)v_1 v_3 & (1-\gamma)v_2 v_3 & \gamma E - \frac{\gamma-1}{2}(2v_3^2 + \mathbf{v}^2) + \left(\frac{5}{3}-\gamma\right)k & \gamma v_3 & \left(\frac{5}{3}-\gamma\right)v_3 & 0 \\ -v_3 k & 0 & 0 & k & 0 & v_3 & 0 \\ -v_3 \omega & 0 & 0 & \omega & 0 & 0 & v_3 \end{pmatrix}.$$

For the RANS-SA equations the fluxes are given by

$F_1 =$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -v_1^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (3-\gamma)v_1 & (1-\gamma)v_2 & (1-\gamma)v_3 & \gamma-1 & 0 \\ -v_1 v_2 & v_2 & v_1 & 0 & 0 & 0 \\ -v_1 v_3 & v_3 & 0 & v_1 & 0 & 0 \\ (\gamma-1)v_1\mathbf{v}^2 - \gamma v_1 E & \gamma E - \frac{\gamma-1}{2}(2v_1^2+\mathbf{v}^2) & (1-\gamma)v_1 v_2 & (1-\gamma)v_1 v_3 & \gamma v_1 & 0 \\ -v_1\tilde{\nu} & \tilde{\nu} & 0 & 0 & 0 & v_1 \end{pmatrix},$$

$F_2 =$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ -v_1 v_2 & v_2 & v_1 & 0 & 0 & 0 \\ -v_2^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (1-\gamma)v_1 & (3-\gamma)v_2 & (1-\gamma)v_3 & \gamma-1 & 0 \\ -v_2 v_3 & 0 & v_3 & v_2 & 0 & 0 \\ (\gamma-1)v_2\mathbf{v}^2 - \gamma v_2 E & (1-\gamma)v_1 v_2 & \gamma E - \frac{\gamma-1}{2}(2v_2^2+\mathbf{v}^2) & (1-\gamma)v_2 v_3 & \gamma v_2 & 0 \\ -v_2\tilde{\nu} & 0 & \tilde{\nu} & 0 & 0 & v_2 \end{pmatrix},$$

$F_3 =$

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ -v_1 v_3 & v_3 & 0 & v_1 & 0 & 0 \\ -v_2 v_3 & 0 & v_3 & v_2 & 0 & 0 \\ -v_3^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (1-\gamma)v_1 & (1-\gamma)v_2 & (3-\gamma)v_3 & \gamma-1 & 0 \\ (\gamma-1)v_3\mathbf{v}^2 - \gamma v_3 E & (1-\gamma)v_1 v_3 & (1-\gamma)v_2 v_3 & \gamma E - \frac{\gamma-1}{2}(2v_3^2+\mathbf{v}^2) & \gamma v_3 & 0 \\ -v_3\tilde{\nu} & 0 & 0 & \tilde{\nu} & 0 & v_3 \end{pmatrix}.$$

The declaration of all variables used here can be found in Chapter 2.

B TauBench

TauBench can be downloaded from the homepage of the 2nd International Workshop on High-Order CFD Methods under

http://www.as.dlr.de/hiocfd/Guideline.html.

It was also introduced in [19]. In addition to that, the following text was taken from the EC funded IDIHOM project [33], courtesy to Tobias Leicht (Tobias.Leicht@dlr.de):

TauBench is a light-weight C code that can be run on a cluster of parallel machines via MPI. It has been developed as a benchmarking tool w.r.t. to CFD codes. It has two parameters that have CFD related names - the number of grid points controls the size of pseudo data arrays per process and the number of time steps controls the number of repetitions of the pseudo code that are used for timing results. Using 250,000 grid points and 10 time steps the code runs for approximately 8 to 15 seconds on typical modern CPUs. The main intention of TauBench was to model the run time behavior of an explicit finite volume code, due to the fact that other available benchmarks like LINPACK were not directly applicable to predict performance of TAU on new machines. Among other things, the pseudo code performs operations that simulate point and face loops as well as ghost point communication via MPI. It was successful in predicting TAU's performance on different parallel machines, in particular the predicted results showed better correlation to TAU run times than other benchmarks.

Using the TauBench run time t_TB as a normalization unit for measured wall clock times t_w, a non-dimensional equivalent sequential run time t_n can be computed as $t_n = n_{CPU} * t_w/t_TB$, where n_{CPU} is the number of parallel processes (which might be larger than the actual number of CPUs due to multi-core CPUs). This non-dimensional time should be considered comparable even on different machines and can be used as a basis for code run time comparisons. The resulting measure might be referred to as "work units".

Notes of caution:

- For best comparability, TauBench should be run on the same number of processes as the actual code in order to include some measure of communication overhead over the network. This yields different numbers for t_TB even on the same machine as the number of processes is varied.

- TauBench should be compiled using the same compiler and compiler settings, in particular optimization flags, as the code to be benchmarked. This must be done manually during configure.
- For codes in other languages than C, the same compiler or settings might not be applicable.
- Users have reported issues compiling/running the code under systems other than Linux.
- Multi threading is not taken into account. It can be included in a crude way assuming linear scaling with the number of threads.
- Other codes and algorithms might require different benchmarking. For strongly implicit schemes, memory bandwidth might be much more important than floating point performance. At DLR we have seen almost a factor of two difference between non-dimensional times for the same case with the same code evaluated on two different machines. Here, a benchmark for linear algebra like LINPACK might be more appropriate.

Evaluation and Recommendation:

- While TauBench is not perfectly fitted for all CFD codes it is probably still the best way of comparing CFD code run time on different machines. A factor of two, which is probably close the maximum uncertainty achieved with this method, might seem large, but such a difference in run time of a CFD code is typically easily achieved with parameter changes for a given case. Thus, the results should not be taken and evaluated as absolute values but rather indicate "orders of magnitude".
- Some care has to be taken to ensure that everyone uses the same parameters (grid points and time steps), compiles the code with appropriate settings and uses the average TauBench run time of several runs on a matching number of processes.

Bibliography

[1] J. D. Anderson et al. *Computational fluid dynamics*, volume 206. McGraw-Hill New York, 1995.

[2] F. Bassi, L. Botti, A. Colombo, A. Ghidoni, and S. Rebay. Implementation of an explicit algebraic reynolds stress model in an implicite very high-order discontinous galerkin solver. In T. Barth, M. Griebel, and D. Keyes, editors, *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2012*, volume 95, pages 111–124. Springer, 2012.

[3] F. Bassi, L. Botti, A. Colombo, D. D. Pietro, and P. Tesini. On the flexibility of agglomeration based physical space discontinuous Galerkin discretizations. *Journal of Computational Physics*, 231(1):45 – 65, 2012.

[4] F. Bassi, L. Botti, A. Colombo, and S. Rebay. Agglomeration based discontinuous Galerkin discretization of the Euler and Navier-Stokes equations. *Computers and Fluids*, 61(0):77 – 85, 2012.

[5] F. Bassi, A. Crivellini, S. Rebay, and M. Savini. Discontinuous Galerkin solution of the Reynolds-averaged Navier-Stokes and k-ω turbulence model equations. *Computers & Fluids*, 34(4-5):507–540, May 2005.

[6] F. Bassi, A. Ghidoni, and S. Rebay. Optimal Runge-Kutta smoothers for the p-multigrid discontinuous Galerkin solution of the 1D Euler equations. *Journal of Computational Physics*, In Press, Corrected Proof:–, 2010.

[7] F. Bassi, A. Ghidoni, S. Rebay, and P. Tesini. High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations. *International Journal for Numerical Methods in Fluids*, 60:847–865, July 2009.

[8] F. Bassi and S. Rebay. A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. *J. Comp. Phys.*, 131:267–279, 1997.

[9] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2d Euler equations. *J. Comp. Phys.*, 138:251–285, 1997.

[10] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini. A high-order accurate discontinuous Finite Element method for inviscid and viscous turbomachinery flows. In R. Decuypere and G. Dibelius, editors, *2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, Antwerpen, Belgium, March 5–7, 1997*, pages 99–108. Technologisch Instituut, 1997.

[11] J. Blazek. *Computational fluid dynamics: principles and applications*. Number Bd. 1 in Referex Engineering. Elsevier, 2001.

[12] Y. A. Blinkov, V. P. Gerdt, and D. Robertz. GINV-project - Gröbner bases constructed by involutive algorithms. Technical report, 2011. http://invo.jinr.ru/ginv/.

[13] M. Ceze and K. J. Fidkowski. Drag prediction using adaptive discontinuous Finite Elements. *Journal of Aircraft*, pages 1–11, Jan. 2014.

[14] L. T. Diosady and D. L. Darmofal. Preconditioning methods for discontinuous Galerkin solutions of the Navier–Stokes equations. *Journal of Computational Physics*, 228(11):3917 – 3935, 2009.

[15] R. Dwight. Robust mesh deformation using the linear elasticity equations. In H. Deconinck and E. Dick, editors, *Computational Fluid Dynamics 2006*, pages 401–406. Springer Berlin Heidelberg, 2009.

[16] D. B. Eisfeld. *Collection of equations for compressible turbulent flow*. Deutsches Zentrum fur Luft- und Raumfahrt e.V. Institut für Aerodynamik und Strömungstechnik Lilienthalplatz 7, Germany, 38108 Braunschweig. ISSN 1614-7790, 2009.

[17] L. C. Evans. Partial differential equations. Graduate studies in mathematics. *American Mathematical Society*, 2, 1998.

[18] D. P. F. Ilinca. Positivity preservation and adaptive solution for the k-w model of turbulence. *AIAA*, J. 36:44 – 50, 1998.

[19] G. Falcone, H. Kredel, M. Krietemeyer, D. Merten, M. Merz, F.-J. Pfreundt, C. Simmendinger, and D. Versick. Integrated performance analysis of computer systems (ipacs). benchmarks for distributed computer systems. *Praxis der Informationsverarbeitung und Kommunikation*, 28(3):150–159, 2005.

[20] M. Feistauer, J. Felcman, and I. Straškraba. *Mathematical and computational methods for compressible flow*. Numerical mathematics and scientific computation. Oxford University Press, 2003.

[21] I. Fejtek. Summary of code validation results for a multiple element airfoil test case. *28th AIAA Fluid Dynamics Conference, AIAA Paper 97-1932*, 1997.

[22] K. Fidkowski. Algebraic tailoring of discontinuous Galerkin p-multigrid for convection. *Computers & Fluids*, 98(0):164 – 176, 2014.

[23] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal. p-multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 207(1):92–113, 2005.

[24] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional Finite Element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79:1309–1331, 2009.

[25] A. Harten and J. M. Hyman. Self adjusting grid methods for one-dimensional hyperbolic conservation laws. *Journal of Computational Physics*, 50(2):235–269, 1983.

[26] R. Hartmann, J. Held, and T. Leicht. Adjoint-based error estimation and adaptive mesh refinement for the RANS and k-ω turbulence model equations. *Journal of Computational Physics*, 230(11):4268–4284, May 2011.

[27] R. Hartmann, J. Held, T. Leicht, and F. Prill. Discontinuous Galerkin methods for computational aerodynamics – 3D adaptive flow simulation with the DLR PADGE code. *Aerospace Science and Technology*, 14(7):512–519, Oct. 2010.

[28] R. Hartmann and T. Leicht. Higher order and adaptive DG methods for compressible flows. In H. Deconinck, editor, *VKI LS 2014-03: 37th Advanced VKI CFD Lecture Series: Recent developments in higher order methods and industrial application in aeronautics, Dec. 9-12, 2013*. Von Karman Institute for Fluid Dynamics, Rhode Saint Genèse, Belgium, 2014.

[29] J. Held and S. Schoenawa. PADGE technical reference of the RANS k-ω turbulence equations. DLR Braunschweig, 2010.

[30] J. L. T. Hiroaki Nishikawa, B. Diskin and D. P. Hammond. Recent advances in agglomerated multigrid. *AIAA-2013-863*, 2013.

[31] C. Hirsch. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*, volume 1. Butterworth-Heinemann, 2007.

[32] C. Klaij, M. van Raalte, H. van der Ven, and J. van der Vegt. h-multigrid for space-time discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 227(2):1024 – 1045, 2007.

[33] N. Kroll. IDIHOM - European project on Industrialization of High-Order Methods for Aeronautical Applications. *ECCOMAS Conference 2012*, 09.2012. Wien, Österreich.

[34] N. Kroll, H. Bieler, H. Deconinck, V. Couaillier, H. van der Ven, and K. Sorensen. *ADIGMA - A European Initiative on the Development of Adaptive Higher Order Variational Methods for Aerospace Applications: Results of a Collaborative Research Project Funded by the European Union, 2006-2009*. Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Springer, 2010.

[35] N. Kroll, C. Hirsch, F. Bassi, C. Johnston, and K. Hillewaert, editors. *IDIHOM - Industrialisation of High-Order Methods - A Top Down Approach*, volume 128 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*. Springer, 2015.

[36] V. I. Krylov. *Approximate calculation of integrals*. Courier Dover Publications, 2006.

[37] B. Landmann, M. Kessler, S. Wagner, and E. Krämer. A parallel, high-order discontinuous Galerkin code for laminar and turbulent flows. *Computers & Fluids*, 37(4):427–438, May 2008.

[38] S. Langer. Point and line implicit methods to improve the efficiency and robustness of the DLR TAU code. In H. P. Kreplin, editor, *New Results in Numerical and Experimental Fluid Mechanics VIII: Contributions to the 17th STAB/DGLR Symposium Berlin, Germany, 2010*, volume 121 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 419–428. Springer, 2010.

[39] S. Langer. Investigation and application of point implicit Runge–Kutta methods to inviscid flow problems. *International Journal for Numerical Methods in Fluids*, pages 69(2): 332–352, 2012.

[40] S. Langer. Application of a line implicit method to fully coupled system of equations for turbulent flow problems. *International Journal of Computational Fluid Dynamics*, 27(3):131–150, 2013.

[41] S. Langer, A. Schwöppe, and N. Kroll. The DLR Flow Solver TAU - Status and Recent Algorithmic Developments. *52nd AIAA Aerospace Sciences Meeting, AIAA Paper 2014-0080*, 2014.

[42] P. D. Lax. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications on Pure and Applied Mathematics*, 7(1):159–193, 1954.

[43] J. Lindenstrauss and L. Tzafriri. Classical banach spaces. *Berlin-New York*, 1977.

[44] H. Luo, J. D. Baum, and R. Löhner. A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. *Journal of Computational Physics*, 211(2):767 – 783, 2006.

[45] H. Luo, J. D. Baum, and R. Löhner. A discontinuous Galerkin method based on a Taylor basis for the compressible flows on arbitrary grids. *Journal of Computational Physics*, 227(20):8875 – 8893, 2008.

[46] B. S. Mascarenhas, B. T. Helenbrook, and H. L. Atkins. Coupling p-multigrid to geometric multigrid for discontinuous Galerkin formulations of the convection-diffusion equation. *Journal of Computational Physics*, 229(10):3664 – 3674, 2010.

[47] D. Mavriplis. Unstructured mesh discretizations and solvers for computational aerodynamics. 18th AIAA Computational Fluid Dynamics Conference, 2007. AIAA 2007-3955.

[48] V. G. Mazia. *Sobolev Spaces: With Applications to Elliptic Partial Differential Equations*, volume 342. Springer, 2011.

[49] I. Moulitsas and G. Karypis. Multilevel algorithms for generating coarse grids for multigrid methods. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, page 45. ACM, 2001.

[50] W. A. Mulder and B. V. Leer. Experiments with implicit upwind methods for the Euler equations. *Journal of Computational Physics*, 59(2):232 – 246, 1985.

[51] C. R. Nastase and D. J. Mavriplis. High-order discontinuous Galerkin methods using an hp-multigrid approach. *Journal of Computational Physics*, 213(1):330 – 357, 2006.

[52] N. C. Nguyen, P.-O. Persson, and J. Peraire. RANS solutions using high order discontinuous Galerkin methods. *AIAA Paper*, 914:2007, 2007.

[53] A. Novotnỳ and I. Straškraba. *Introduction to the mathematical theory of compressible flow*. Oxford lecture series in mathematics and its applications. Oxford University Press, 2004.

[54] H.-O. Peitgen. *Newton's method and dynamical systems*. Springer, 1989.

[55] L. Prandtl, H. Oertel, and M. Böhle. *Prandtl-Führer durch die Strömungslehre: Grundlagen und Phänomene*. Vieweg, 2002.

[56] F. Prill. *Diskontinuierliche Galerkin-Methoden und schnelle iterative Löser zur Simulation kompressibler Strömungen*. PhD thesis, TU Hamburg-Harburg, 2010. available as DLR-FB 2010-17, ISSN 1434-8454.

[57] A. S. R., J. F. T., and P. R. Spalart. Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model. *ICCFD7*, 1902, 2012.

[58] W. H. Reed and T. Hill. Triangular mesh methods for the neutron transport equation. *Los Alamos Report LA-UR-73-479*, 1973.

[59] P. L. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, Oct. 1981.

[60] H. Rosenbrock. Some general implicit processes for the numerical solution of differential equations. *The Computer Journal*, 5(4):329–330, 1963.

[61] C. Rumsey, J. Slotnick, M. Long, R. Stuever, and T. Wayman. Summary of the first AIAA CFD high-lift prediction workshop. *Journal of Aircraft*, 48(6):2068–2079, 2011.

[62] V. V. Rusanov. Calculation of the interaction of unsteady shock waves with obstacles. *Z. Vycisl. Mat. i Mat. Fiz.(J. Comput. Math. and Math. Phys.)*, 1(2):267–279, 1961. in Russian.

[63] Y. Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[64] S. Schoenawa and R. Hartmann. Discontinuous Galerkin discretization of the Reynolds-averaged Navier–Stokes equations with the shear-stress transport model. *Journal of Computational Physics*, 262:194 – 216, 2014.

[65] K. Shahbazi, D. J. Mavriplis, and N. K. Burgess. Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations. *Journal of Computational Physics*, 228(21):7917 – 7940, 2009.

[66] P. Spalart and S. Allmaras. One-equation turbulence model for aerodynamic flows. *Recherche aerospatiale*, (1):5–21, 1994.

[67] W. Sutherland. The viscosity of gases and molecular force. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 36(223):507–531, 1893.

[68] P. Tesini. An h-multigrid approach for high-order discountinuous Galerkin methods, (doctoral dissertation). 2008-05-08. Retrieved from http://hdl.handle.net/10446/52.

[69] L. Thomas. Elliptic problems in linear difference equations over a network. *Watson Sci. Comput. Lab. Rept., Columbia University, New York*, 1949.

[70] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[71] J. van der Vegt and S. Rhebergen. HP-Multigrid as smoother algorithm for higher order discontinuous Galerkin discretizations of advection dominated flows: Part I. multilevel analysis. *Journal of Computational Physics*, 231(22):7537 – 7563, 2012.

[72] J. van der Vegt and S. Rhebergen. HP-Multigrid as smoother algorithm for higher order discontinuous Galerkin discretizations of advection dominated flows. Part II: Optimization of the Runge–Kutta smoother. *Journal of Computational Physics*, 231(22):7564 – 7583, 2012.

[73] M. Wallraff, R. Hartmann, and T. Leicht. Multigrid solver algorithms for DG methods and applications to aerodynamic flows. In N. Kroll, C. Hirsch, F. Bassi, C. Johnston, and K. Hillewaert, editors, *IDIHOM - Industrialisation of High-Order Methods - A Top Down Approach*, volume 128 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 153–178. Springer, 2015.

[74] M. Wallraff and T. Leicht. 3D application of higher order multigrid algorithms for a RANS-$k\omega$ DG-solver. In R. Abgrall, H. Beaugendre, P. M. Congedo, C. Dobrzynski, V. Perrier, and M. Ricchiuto, editors, *High Order Nonlinear Numerical Schemes for Evolutionary PDEs - Proceedings of the European Workshop HONOM 2013 , 18-22nd March, Bordeaux*, volume 99 of *Lecture Notes in Computational Science and Engineering*, pages 77–88. Springer, 2014.

[75] M. Wallraff and T. Leicht. Higher order multigrid algorithms for a discontinuous Galerkin RANS solver. *52nd AIAA Aerospace Sciences Meeting, AIAA Paper 2014-0936*, 2014.

[76] M. Wallraff, T. Leicht, and M. Lange-Hegermann. Numerical flux functions for Reynolds-averaged Navier-Stokes and $k\omega$ turbulence model computations with a line-preconditioned p-multigrid discontinuous Galerkin solver. *International Journal for Numerical Methods in Fluids*, 71(8):1055–1072, 2013.

[77] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh, N. Kroll, G. May, P.-O. Persson, B. van Leer, and M. Visbal. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, pages 72(8): 811–845, 2013.

[78] D. C. Wilcox. Reassessment of the scale-determining equation for advanced turbulence models. *AIAA Journal*, 26(11):1299–1310, 1988.

[79] D. C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, Inc., La Canada CA, 1993.

[80] A. Wolkov, C. Hirsch, and B. Leonard. Discontinuous Galerkin method on unstructured hexahedral grids for 3D Euler and Navier-Stokes equations. *AIAA*, 4078:2007, 2007.